CALIFORNIA STATE UNIVERSITY NORTHRIDGE

INTRODUCTION TO BASIC PROGRAMMING: A STUDENT WORKBOOK FOR THE ATARI

A graduate project submitted in partial satisfaction of the requirements for the degree of Master of Arts in

Secondary Education

by

Jon M. Lauritzen

May, 1985

The Graduate Project of Jon. M. Lauritzen is approved:

Barnabas B. Hughes, Ph. D.

Christine C. Smith, Ed. D.

Charles H. Heimler, Ed. D., Chair

California State University, Northridge

TABLE OF CONTENTS

ABSTRACT

AN INTRODUCTION TO BASIC PROGRAMMING:
STUDENT WORKBOOK FEATURING THE ATARI

by

Jon M. Lauritzen

Master of Arts in Secondary Education

With the advent of the microcomputer, decisions were difficult, there were a wide variety of hardware choices, and a limited supply of material suitable for classroom use. Those who instructed computer classes had to collect materials from many sources, then write their own lessons. In that historical context, the workbook in this project was developed.

Individual lessons were written and compiled into a workbook, for the instruction of ATARI BASIC. The material has been used and tested in a variety of classrooms, then revised to provide an instructional guide, or reference document. Each lesson presents the instructor with the materials necessary to provide information and answers for a lecture, demonstration, or discussion. The student, working on the computer explores each of the areas discussed in a lesson. Exercises then direct students through each section of BASIC programming, where they expand upon the concepts presented.

The workbook begins with an introduction to the keyboard and screen editing. Lesson two introduces the concepts of BASIC programming statements and commands, then explores the treatment of mathematical functions on the computer. The third lesson introduces program planning, and the use of computer peripherals, including the disk drive and the printer. In lesson four the student will expand his or her understanding of programming features. Lesson five looks at the BASIC concepts of branching and loops. The treatment of variables in BASIC is developed in lesson six, and lesson seven explores functions programmed into the computer. In the concluding lesson, the student is introduced to the use of sound, color, and graphics in programming. Throughout the workbook there is an attempt to allow the student to discover as many of the programming concepts as possible.

CHAPTER I
INTRODUCTION

When the computer was first introduced to the classroom, educators found little or no instructional material acceptable for use with their instructional program. Teachers, whether they had been assigned, or had chosen to work with computers because of a personal interest, were frustrated by the limited quantity of quality material suitable for classroom use. To be effective; therefore, it became necessary for educators to create their own materials, including lessons and support documents for use with the computer.

To create their own materials required a great deal of research through many different sources in order for the teachers to educate themselves in the operation of complicated hardware and acquire necessary programming skills. This research was required of teachers to the extent that they could not only operate the computer effectively, but could also structure a viable programming lesson for use with their students. Those who were fortunate enough to find courses either through a college, university or commercial organization had certain advantages in understanding BASIC programming and the operation of the computer.

Teachers who had a math or science background, usually had some experience in working with computers. These teachers met their assignment with experience and enthusiasm, eagerly designing and implementing their assigned course. Those not already computer literate, however, were likely to become easily discouraged (Bell, 1983).

The history of using computers in the schools is one of changing trends. Use of computers in the classroom started sometime in the sixties. Instructional computing began with a reliance on the mainframe computers of certain large universities, since they were the only ones available at the time. With the advent of the microcomputer, specialized retail outlets began to offer programming courses, as did many schools and colleges (DeDault, Harvey, 1983).

Initially, there was little or no instructional software available for the microcomputer. What followed was a virtual avalanche of programs and printed material, making the selection process for the teacher exceedingly difficult. The many magazines and volumes of books provided a wealth of material. Much of this material, however, was unsuitable or too technical for classroom use.

The Problem

In the early years of computer education, an educator required to use a microcomputer for classroom instruction could get very little help. Few materials were available, and each individual teacher was forced to resort to his or her own resources for computer related lessons. Exploring the varying functions and capabilities of a given microcomputer and the programming languages as they also became available, was quite a challenge. This was extremely difficult, particularly if the instructor had little or no experience with the computer selected for classroom use. Moreover, there has always been a shortage of courses for microcomputers, particularly courses which offered the student a "hands-on" experience. Locating programming courses, for many of the different brands of microcomputers, unless offered by a retail outlet was next to impossible. Finding programming materials for a specific type of hardware, was equally difficult. Here to, when material was available, it was frequently of a technical nature, difficult to understand and impossible to interpret. Other materials were poorly organized, with a structure which would not adapt well to a classroom situation.

Need For The Project

Due to the lack of specific material for teaching BASIC on the ATARI computer, it was necessary to compile a series of lessons and exercises to provide instruction and activities for assigned classes. Since there is still a shortage of materials in the area of specialized hardware oriented programming, the need exists for a workbook which is suitable for classroom use.

The ATARI microcomputer with its modified BASIC language, and the additional features it offers, is one of those microcomputers which require different programming techniques to take full advantage of its special capabilities. In addition as the use of peripherals, such as disk drives, and printers becomes more wide spread, students need to be familiar with their operation. Features such as color, sound, and graphics, give the ATARI, and some of the other microcomputers, a motivational appeal which should be incorporated into programming courses.

General textbooks for microcomputers have a tendency to be incomplete or inadequate, and have limited use in the classroom. Standard textbooks on BASIC programming are at variance with ATARI BASIC and often lead to frustration for the student. Programming textbooks which do not meet the students needs are of little help in learning to use the graphics capability of the ATARI (Haskell, 1983).

A student attempting to work individually found a void in effective materials for self-instruction. The teacher who attempted to give individual attention to students working, "hands-on", at a computer station, was frequently frustrated both in the attempt to give assistance, and in maintaining discipline among other students in the class. Those students not at a computer station, who had insufficient material to work with or to keep them occupied, often become disinterested. Good activity related study material is essential in this area.

Initially most teachers have a very limited number of computers to work with, perhaps only one or two. The "hands-on" approach to learning was almost impossible to conceive, let alone implement. Even in the computer labs of today where microcomputers have proliferated to a great extent, there are cases where a teacher is beginning a new program with an insufficient number of computers for each individual student to have access to one whenever it is needed. It does, therefore, become necessary to provide activities for those students who are not able to work at a computer station during a class period. The teacher new to computers, seldom has the experience to plan effective "seat work" activities for a computer class, even many of those with extensive computer experience, were limited in this area which requires a special skill.

Historically the computer teacher has run into difficulty when attempting to piece together a lesson from the inadequate documentation available with most computers. Unfortunately the documentation which was available was of a technical nature and was often difficult to interpret. It therefore became necessary for the author in teaching his computer classes to begin the long and arduous task of collecting useful ideas from all available sources. These sources included the product documentation, the few books and magazines available, and consultation with other experienced individuals. Using these resources it was possible to piece together a workable lesson for use with the author's students. A few experienced individuals were available to provide assistance, these have been extremely helpful: Dr. Sid Kolpas, of TRS, Bill Gibbons of HW computers, Mike Seldon, a self educated expert on the ATARI, and Dr. Charles Heimler of California State University at Northridge.

Purpose of the Project

It was the purpose of this project to develop a self-directing student workbook for classroom or individual use. The use of such a workbook, with its individual lessons, will free the instructor to provide assistance to those students who need individual attention with either computer hardware or programming techniques. At the same time it will provide those students who do not have access to computer learning stations, activities to occupy and challenge them.

The workbook designed for this project will fill the needs of ATARI computer teachers and students as they pursue the study of BASIC programming. The teacher is provided with a study and activity guide. While the student using this workbook will, with some limited direction, be able to work at the computer, they will also discover the intricacies of the hardware and the fascination of program development. Further, the workbook will free the teacher from the burden of daily preparing viable lessons for classroom use. Also access to this material will allow more time for work with the individual student and provide constructive in-class activities for students when they are not able to use a computer.

Definition of Terms

Workbook

As an attempt is made to present the material in this project there must be an understanding of terms. When the term Workbook is used in the context of this project it is defined as a study document which can be used by an individual working to comprehend the concepts presented. It is also the type of document which can be used by an instructor who can present the material for exploration by a group of students. In either event the document may be used as a primary study guide or as a supplementary resource document.

Programming

The use of computers in instruction has introduced many misunderstood terms. Many of those with little understanding of instruction in the realm of computers, might relate the term Programming to the entire structure of course work within the area of related computer studies. A more precise definition of this concept is necessary as it applies to the material presented here. Programming is the process of comprehending a problem, planning the structure of the solution, writing a program, and then correcting all errors so the program will itself, then present a solution. Programming requires the use of a specific language for communication between the computer and the programmer.

BASIC

One of the most widespread and useful languages is BASIC, an acronym for Beginner's All-Purpose Symbolic Instruction Code. BASIC is a particularly useful language for use with students, as it is closer to the English language than the others. BASIC also allows many creative variations in the approach to a programming problem.

Computer Lab

A Computer Lab for purposes of this project is defined as a microcomputer laboratory. The lab may be any room or facility set up with one or more computers, and designed for use with the instruction of computer programming or computer related subjects. An ideal lab would contain a sufficient number of computer stations to accommodate the anticipated enrollment. Each station should contain a microcomputer with an external storage devise such as a disk drive, a monitor with sound and full color capability, and access to a printer. In addition their should be a seating area where students can receive instructions or work away from the distraction of the computers.

Outline of the Project

Chapter II reviews the literature related to the use of BASIC as a computer language for the classroom, the acquitision of programming knowledge and skills by an individual instructor, the learning theory involved in the instruction of students in a computer lab situation, and the need for student instructional material for the teaching of ATARI BASIC

Chapter III details the writing of the workbook, constructed for this project. It explains and documents the development of the materials which were created and assembled within the structure of a workbook. The presentation of the activities included here, is designed to give an overview of the material which enables the user to judge its validity as it relates to the selection of study needs. The difficulty in locating materials for use with the instructional program, a constant in the teaching of any computer related subject, is presented. The problems involved in adapting materials for use with the ATARI are detailed. Finally the studies of the techniques of learning which lead to the use of discovery procedures in the learning process, wherever possible, is explored and documented.

Chapter IV includes the description of the actual workbook as the primary objective of this project. The outline of the eight lessons gives a description of each, as it is related to the overall project. Then, each lesson is detailed as its description is documented

Chapter V documents the classroom use and evaluation of the workbook material. The documentation procedures are outlined and used to project the use of the materials as individual lessons, or activities. The possible uses of the complete workbook are analyzed and discussed. Development of materials and studies related to the project have only begun to cover the subject area. There is a strong indication that there is still significant need in the area of lesson and activity development. Future needs are in the computer fields of program instruction, and the use of software in the instructional program.

CHAPTER II
REVIEW OF THE LITERATURE

Even though the debate over which programming language should be taught in the classroom continues to rage, there are valid arguments for the teaching of BASIC. For most teachers the question of which language they will use has already been decided, either by the school district, a school administrator or the department of which they are a part. For the teachers, who will be teaching BASIC it becomes a question of where to obtain the necessary training. The teacher must select the materials to be used in the classroom with the computer. When unable to locate or adopt appropriate materials the teacher must then write the lessons and worksheets necessary to provide information and activities for their classes.

Programming in BASIC

Perhaps the most eloquent defender for the teaching of BASIC in the classroom is Arthur Luehrmann, founding partner of Computer Literacy in Berkeley, Ca. According to Luehrman:

> There is no "right" computer language. And preoccupation with languages is misleading because it takes us away from the important problems: teaching people to think carefully and write clearly for the computer (Luehrmann, 1982, p.23).

Luehrman (1982) also points out that BASIC is the common language of the computer, and that all microcomputers have a version of BASIC. The important thing then is to teach, "clear, readable, programs" in the common language of BASIC (p. 24).

Learning BASIC

Learning to program, according to Electronic Learning, in "The Computer Primer", is simply learning to "talk" to the computer. BASIC is the language most widely used for instructional applications.

> Like any language, a computer language consists of both a vocabulary and a set of grammatical rules. Unlike foreign languages, though, a computer language, such as BASIC consists of a limited number of standard English words or abbreviations, a set of symbols such as those used in mathematics, and just a few rules of grammar (<u>Electronic Learning</u>, 1982, p. 20).

In, "Is a Computer Hard to Learn", from <u>Consumer Reports</u>, there is support for the concept that learning a computer language is a procedure which because it is similar to learning any language, need not create difficulty for the learner. It is also pointed out that one need not learn the entire language in order to gain value from it.

> Learning BASIC, or another programming language, is an enterprise not unlike learning French. You don't need to learn programming to use a computer, just as you don't need to learn French to vacation in Quebec. But at least a little programming is helpful in understanding how the computer works and in figuring out what you may have done wrong (Consumer Reports, 1983, p. 488).

With just a few BASIC commands students can begin seeing the results of their programming efforts almost immediately. It is further pointed out by Paul Bonner in Personal Computing, that since BASIC is an interpreted rather than compiled language, it has the advantage of immediate execution (Bonner, 1983, p. 128).

Learning Theory

When The language question has been settled, the educator must look at learning theory to determine the method which will best communicate the capabilities of the computer and the language to the student. The argument for the use of computers in the classroom should be looked at in an attempt to avoid those abuses, computer instructors have been accused of in the past.

In an interview in the "People in Computing" section of Personal Computing magazine, Richard Koff of the Northwestern University in Chicago, states: "there's no better way to learn than by hands-on experience." Koff goes on to point out how he has set up elaborate computer "what-if" models to teach his students (Personal Computing, 1983, p. 15).

When Paul Bonner quotes Nels Winkless of the Excalibur Company, Winkless would seem to support the explorative nature of learning to program as he indicates that a traditional analytical approach to programming:

> virtually excludes intuitive learners...The traditional educational route has been strictly analytical-you sit down and get a little bit of introduction,...First you learn a set of rules, and then you sit there and work through those rules,... That certainly impedes learning for those not inclined to work on a rule-driven system (Bonner, 1983, p. 137).

Another supporter of the discovery theory of learning to program is Tony Morris of the University of Michigan, as he relates in an interview about teaching his own children to work with the computer. His goal was to have the children understand the computer and how it worked. To accomplish this he banned all commercial software from the home. "If the kids had used only commercial software from the start, they wouldn't have had any curiosity. It would have been too easy." (Frenkel, 1983, p. 45)

Decker F. Walker of Stanford, relates his philosophy of learning with the microcomputer:

> My experience - derived from three years of reading, thinking, and working with computers and computer-based education programs - leads me to identify seven main ways that today's microcomputers can contribute to education.

> These are: 1)more active learning, 2) more varied sensory and conceptual modes, 3) less mental drudgery, 4) learning nearer the speed of thought, 5) learning better tailored to individuals, 6) more independent learning, and 7) better aids to abstraction (Walker, 1983, p. 103).

T.H. Bell, Secretary U.S. department of Education writes about the spread of microcomputers and the need for quality educational materials for use with them, even though he speaks of software development these ideas translate easily to other areas of microcomputer education.

> What we need is a major effort to develop some super software packages that will: 1) Motivate the student through reaching into the interests and concerns of the minds of the learners; 2)Branch out and present the subject matter again and again to the learner who did not grasp the concept the first time it was presented; 3) Move ahead rapidly with the gifted and talented learners; 4)present the subject matter with the utmost in attractive sound, color and animation; 5) Reinforce the students' desire to learn more by offering prompts, cues and encouragement in working through some of the most crucial phases of some lessons; and 6) Keep careful tabulation of each student's progress, correct and erroneous responses, and print the same out for use of the teacher (Bell, 1984, p. 81).

In planning a program, writing materials, or just locating ideas, it is important to view the computer as a tool which can allow the student a great deal of flexibility in learning, but the learning must also be accompanied by valid instructional techniques.

Obtaining Programming Skills

The Instructor must first aquire skills in programming, before these can be taught to others. Initially there were few resources for obtaining these skills. More recently, however many sources of instruction have become available. Some are exotic and some mundane, Leroy Finkel, author, educator, and advisor, when interviewed by Lorraine Hopping, assistant editor to Teaching and Computing, suggests that if one is going to be working with a specific type of computer, he or she should first take a short course that is specific to that machine, he cautions that those who really want to learn programming can expect to invest a lot of time. She quotes Finkel as saying:

> In a short course, you can learn a little programming, computer operations, and how to run software. You have to invest a minimum of 12 weeks, probably more to become proficient in programming (Hopping, 1983, p. 56).

Hopping (1983) has evaluated eight alternative sources of computer education including: 1)Computer stores, where courses are offered, but don't expect much from the sales people she cautions, 2) Computer fairs, particularly those which have begun to "specialize by machine type,..., thereby offering more in-depth coverage of one machine", 3) Resorts, it is possible to educate the whole family while vacationing, 4) Nonprofit organizations such as scout troups or clubs, 5) Universities where many in-depth courses are available, 6) Adult education centers, 7) Computer camps, and 8) the manufacturer, who offer computer users various programs of instruction specific to their machines (Pp. 56-59)

Add to this list the user groups which can be an invaluable source of both general and specific programming information. For the teacher there is now a wealth of in-service courses, also a certain amount of on the job training is available for those currently beginning their education in computers.

The Workbooks for Student Use

When the type of computer has been determined, the programming language has been selected, and the instructor has been prepared with adequate programming knowledge, there is still the question of selecting the materials for use with the course to be taught.
The textbooks and related materials available during the development of this project, while offering a valuable source of information were lacking in many respects. Some of the materials were more technical than an introductory course required, this was true of Cummins' BASIC Programming, others were either too general or too specific for this type of course (Cummins, Kuechmann, 1983). The format used by TRS in its BASIC student manuals was one of

the best at the time, but there were several problems. First they require an instructor to give the information to complete the lessons, second the programming techniques were very elementary, and third there is no color or sound and very limited graphics capabilities (Radio Shack, 1979).

Other volumes such as Albrecht's ATARI BASIC are good references, but are written at an advanced level and would not be effective at the secondary level (Albrecht, Finkel, Brown, 1979).

Use of documentation or available instructional material has proven ineffective, as is indicated by Dianne Martin of George Washington University, when she says:

> A walk through the typical local library will reveal a small selection of out-of-date books on computers and data processing comparable to reading a book about the Model T to study automobiles...the local bookstore can leave...[one]...confused and overwhelmed by the diversity and sheer number of trade books available on the subject. Few books are written with the general adult learner in mind (Martin, Heller, 1984, p. 125).

With a lack of materials which suit the needs of the classroom instructor, the only choice remaining seemed to write a workbook in the format which has historically proven effective as a means of instructing and giving the student a means for recording responses.

Summary

In reviewing the literature for this project, conclusions have been drawn in five related areas: (a) BASIC language programming should be taught in the schools, (b) an individual educator can find means of developing programming skills, becoming proficient enough to work in a computer equipped classroom, (c) it is possible to conduct a class implementing a theory of learning in which the student is allowed to discover many of the desirable programming skills, and (d) a workbook format would be the most effective medium for this project.

CHAPTER III
WRITING THE WORKBOOK

The workbook developed in this project, was produced in response to the lack of adequate materials available for instructional purposes. Teaching ATARI BASIC courses required specialized programs, techniques, and lessons, none of which were available in a form designed for classroom use. additional problems included difficulty in obtaining instruction, a need for organized, activity oriented, lessons, and the difficulty of translating technical references into a usable form. The only solution to the problems which had been encountered it seemed would be to write a detailed set of lessons which would challange and motivate, as well as educate the student. The lessons were written, tested, revised, and eventually compiled into a workbook format.

Design

A working knowledge of the BASIC programming language had been acquired by the author through the completion of several courses in programming on the TRS 80 microcomputer.

The ATARI computer, however, required an understanding of the existing variations in both the hardware and in the programming so as to take advantage of its special features. To understand, and subsequently teach, these specialized features required much research and experimentation. Fortunately, resource people were available to give assistance at various times during the development of the workbook.

The author determined the best approach to the teaching of BASIC was to introduce statements and commands in a specific sequence. But first the student needed to be taught the special features of the ATARI keyboard, then shown or allowed to discover the differences in programming the ATARI computer, as these differences were encountered. Finally those special features of the ATARI ROM (read only memory), which were not available on other computers were to be explored and developed, as program components.

Development of Learning Activities

Activities which would keep the student interested and challenged were essential to each stage of a computer class. The student appears to gain most from activities when they lead to the discovery of general concepts as the problem is explored. Ideally the student would have access to a computer at any time during a class period; however, this isn't always possible due to a limited number of computers available under most circumstances. It was necessary, therefore, to provide activities in the workbook which can be completed without the aid of the computer. Activities of this nature are effective when directed by the instructor, but they must be structured in such a way that they can be completed by a student working individually. Additional activities in the workbook have been designed as "seat work", for exploration when a computer isn't available. Verification of the results of this type of work is still required by entering the problems or programs on the computer and comparing the output.

Locating and Evaluating Materials

While much of the material included in the workbook, is the result of personal experience and experimentation, several sources proved invaluable in its development: 1) the <u>TRS Student Manual, Vols. I and II</u> (Radio Shack Corp., 1979). 2) <u>An Invitation to Programming</u>, (ATARI Corp., 1981). 3) <u>ATARI BASIC</u>, (Albrecht, Finkel, and Brown).

There are numerous publications featuring BASIC programming, sorting through these materials, made it possible to find ideas which would work in a classroom situation. The initial problem was one of locating sufficient material adaptable to a programming class featuring ATARI hardware. When a flood of material began to emerge, it became a matter of sifting through quantities of avialable magazines, books and hardware documentation, in an attempt to keep current on programming ideas.

Adapting materials for use with the ATARI

In order to adapt materials for the ATARI it became necessary to test all general materials to see what was compatible with ATARI BASIC. Several areas which caused difficulty when attempting to adapt from materials written for general use included: TAB statements, ATARI uses a POSITION (POS.) statement to replace TAB, VTAB, and HTAB; DIMENSION statements are required with all string variables on the ATARI, but may not be necessary on other computers; String and subscripted variables are both dealt with differently on the ATARI than on most other computers. Where the instructor has experience with other computers it is sometimes effective to compare the various means used by each type of computer, for the benefit of the class. If programs or commands require adaptation to function properly on different computers, the activities for which they are intended should include this fact. With most of the simpler programming ideas, generally adapted from early resources, it was merely a matter of changing one or two program lines or perhaps only one or two symbols, to accomplish the desired result.

Ideas for interesting and educational activities for use in the workbook were derived from several sources: Compute! and Antic magazines were of particular value with ideas for use with the ATARI. The Volumes published by Compute! featuring articles written specifically for the ATARI were of value in developing materials in the areas of color, sound, and graphics where there were very limited resources in other publications (Compute!, 1981). Another valuable resource in developing and clarifying concepts in programming was Lon Poole's, Your ATARI Computer. The sections dealing with strings, arrays, and ATARI peripherals, contained in-depth studies of these areas (Poole, McNiff, Cook, 1982).

Techniques of Learning

Among the activities incorporated within the workbook format are several learning techniques. Lessons are: 1) Tutorial - information is given as text, or demonstrated by the instructor; 2) Discovery - the student is directed to complete specific procedures and record the results; 3) Evaluative - problems are assigned with instructions to check and evaluate the results on the computer, or programs with deliberate errors may be included with instructions to "de-bug" the program (remove the errors); 4) Practice - exercises are included for the student to complete, either with the computer, at a desk or both; and 5) Experimental - programming assignments are given and the student is to write or complete a segment or an entire program, documenting each step in the program, and explaining the RUN (output) of the program. Each of these techniques is designed to place the emphasis on the discovery of programming procedures by the student as lessons are completed, how well this is implemented would depend on the instructor of the course.

Summary

In this chapter the need for a workbook which would aid in the instruction of ATARI BASIC was described. The design of the workbook was reviewed; the types of activities, the reasoning behind the structure of the activities, and their purpose in the instruction was presented. The procedure for locating and evaluating materials along with the process of adapting materials for the ATARI was outlined. Finally, techniques of learning incorporated into the workbook were explored.

Collecting, organizing, testing, and writing the material for this workbook was initiated to provide lessons and activities, for classes being taught by the author. The material as compiled in the workbook, is presented as a resource for those who will teach BASIC programming on the ATARI.

CHAPTER IV

DESCRIPTION OF THE WORKBOOK

In organizing the material for this workbook it was necessary to consider who might use it, and how to most effectively arrange it for a variety of users. Initially the material was designed for use by an instructor with a group of students. Later it was determined that the material could also be used effectively by a student working individually. It is necessary, therefore, to provide instructions for use of the workbook, for both the instructor and the student. The most effective methods of using the individual lessons and the workbook as a whole have been presented here. Introductory material includes objectives for each lesson.

The lessons are designed as "units" of instruction, and may take a week for a class in a secondary school, several hours with adults in an in-service situation, or an individual may complete each section more rapidly working alone. In each case, however, it is necessary for the student to spend a certain amount of time individually practicing the skills presented. This summary of the workbook, will be presented as if it were to be used in a classroom setting with an instructor directing the lessons.

It includes the following:

- Introduction
- Lesson I: Introduction to the microcomputer
  - Computer definitions
  - Hardware and peripheral hook-up
  - The keyboard
  - Keyboard editing
- Lesson II: Meeting ATARI BASIC
  - The PRINT command
  - Mathematical operations
  - Mathematical exercises
  - Algebra on the ATARI
  - RUN, LIST and NEW commands
- Lesson III: Steps in planning, saving and printing a program
  - The flow chart
  - Program analysis
  - Disk drive
    - Operation
    - Format
  - Printer operation
- Lesson IV: Additional programming features
  - LET,END, STOP, AND CONT commands
  - BASIC message responses
  - The INPUT statement
  - READ/DATA, RESTORE statements

Lesson V: Branching and loops

- Unconditional branching: the GOTO statement
- Conditional branching: IF/THEN statements
- FOR/NEXT loops
- Subroutines: the GOSUB/RETURN statements

Lesson VI: variables

- Definitions and limitations
- Assignment of variables
    - Numeric
    - String
- DIMENSION statement
- Subscripted variables
    - Array
    - Matrix

Lesson VII: Computer functions

- ASCII (ASC)
- CHARACTER STRING (CHR$)
- LENGTH (LEN)
- RANDOM (RND)

Mathematical and scientific functions

- SQUARE ROOT (SQR)
- LOG, SIN, and COS

Lesson VIII: Sound, color and graphics

- SOUND statement
- COLOR statement
- Graphic modes

Evaluation and summary

LESSON I: Introduction to the Microcomputer

The instructor will begin with an introduction to the computer hardware and peripherals. The student learns about the computer components and how to "hook-up" (assemble) them. When the equipment is together, and the power is turned on, the student begins exploring the keyboard. The differences between a typewriter and a computer keyboard is evaluated. The keyboard graphics of the ATARI are presented and explored for the enjoyment and interest of the student. The student should become comfortable with the keyboard and its special functions, particularly the editing features which allow the correction or changing of anything on the screen without retyping entire segments of material. This lesson is concluded with a practical quiz, in which the student performs given operations on the computer, under the observation of the instructor.

LESSON II: Meeting ATARI BASIC

In Lesson 2 the instructor can begin the introduction of BASIC language commands. PRINT is the first command presented, the student is given lines of material to type, leading to the discovery of the variations and limitations of this important command. After the introduction of the PRINT command the fundamental math operations of BASIC are introduced, the student is asked to compute simplified math problems, then type them in to the computer in order to see how the computer deals with the operations of addition, subtraction, multiplication, division, and exponents. The proper grouping of mathematical operations is explored, and the treatment of Algebraic expressions in BASIC is demonstrated. Next, in displaying large and small numbers, scientific notation is compared to the computer form of "E-Notation". The student is given the first in a series of programs, for exploration and development. Definitions previously encountered as the lesson developed are: BASIC, bit, byte, RAM, and ROM. The statements REM and PRINT, and commands RUN, LIST, and NEW also previously introduced are reviewed to summarize this lesson.

LESSON III: Steps in Planning, Saving, and Printing a Program

This lesson leads the student into the area of BASIC program development. The design of a flow chart is presented for student use in outlining the scope and structure of a program. The actual writing of a program is accomplished by following the step by step procedures outlined: 1) analyzing the problem, 2) flow charting the solution, 3) writing the program, 4) Typing the program into the computer, and 5) RUNning (executing), and debugging (correcting errors) the program. The student then analyzes the program for desired results. The operation of the disk drive and printer are to be demonstrated and discussed. A diskette will be formatted, (ie) prepared for saving programs, sample programs will be saved on the diskette, then printed on paper. The instructor would critique the printed material, to summarize the lesson, and evaluate student progress.

LESSON IV: Additional Programming Features

The additional commands LET, END, STOP, and CONT are featured in this lesson. "Debugging" is pursued in a series of programs where each is to be rewritten to produce a desired result. BASIC message responses of the ATARI are reviewed, as three types of errors are discussed, along with informative messages, called screen prompts. The INPUT statement is explored as another means of entering data into the computer either as single or multiple items. An additional form of entering data, the READ/DATA statement, is most useful when a great deal of information must be entered. The RESTORE statement, and its relation to READ/DATA is explored, as it allows the reuse of previous data.

LESSON V: Branching and Loops

Two aspects of branching are introduced in this lesson. In unconditional branching the computer is not given an alternative, but must turn over control of the program to another section. The GOTO statement is employed for this programming situation. Conditional branching in which a given condition or set of conditions must be met includes the IF/THEN statement. GOTO and IF/THEN statements can be used to set up loops in a program, where a part of the program is to be repeated. The FOR/NEXT statement, however, always creates a loop, with the number of times the loop is to be repeated defined in the statement. Various aspects of these statements are explored as the student works through the exercises of this lesson. A subroutine allows the reuse of any segment of the program which would be needed more than once. The GOSUB/RETURN statements are introduced and expanded in this lesson.

LESSON VI: Variables

The assignment of values to numeric variables is accomplished through the let statement presented in a previous lesson. The limitations and scope of these variables is presented here. The use and assignment of string variables is much more complex and requires a more in depth study. The Dimensioning requirements of the ATARI, necessary to deal with string variable needs to be presented at the same time as these variables are discussed, in order for the student to use them in programs. Subscripted variables on the ATARI involve a considerable variation from other forms of BASIC and these differences must also be explored. Both arrays and the matrix are introduced in this lesson.

LESSON VII: Computer Functions

The most useful of the computer, mathematical and scientific functions programmed into BASIC, are discussed in this lesson. Two purposes are accomplished by the introduction of computer functions through the manipulation of string variables, first it can be the vehicle for the introduction of several commands, (ASC) which will give the ASCII code (American Standard Code for Information Interchange), for any symbol, (CHR$) produces the Character string for an ASCII number, and (LEN) will give the length of a particular string. The other feature of this exercise is the development of an understanding of the use of the

ASCII code. Random (RND) allows a random number to be selected, when used with the integer (INT) function, the number can be defined within specific limits. The number can then be used in a program, for example the numbers between 1 and 6 which could be used to simulate the throw of a dice, are used in an exercise to illustrate this function. Other functions explored here are: (ABS) for the absolute value, (SQR) Square root, (LOG) will give the logarithmic value, (SIN) sine, and (COS) cosine return those mathematical values.

LESSON VIII: Sound, Color, and Graphics

The introduction of sound, color and graphics at this time in the program, is to add interest and motivate the student to use the programming techniques learned up to this point in the course. The sound portion of the lesson will examine voice, pitch, note, and volume within the SOUND statement, exercises will emphasize sound effects and music. Color will be introduced by the use of The SETCOLOR statement, including use of color registers, color codes, and luminance. The graphics modes, including text and graphics will be included in the exercises which will display the various screens, colors, and size of text.

Evaluation and Summary

Sample testing instruments will be included and may be used as a summarizing or evaluative technique. These testing instruments will include both written questions and problems in computer programming, and practical exercise problems for demonstration of computer skills.

CHAPTER V

CLASSROOM USE OF THE WORKBOOK

The various lessons of the workbook have been used at every level from the upper grades in an elementary school, to in-service classes with adults. At the elementary level several of the lessons were presented to students of the Chatsworth Park Elementary School Computer Club, The response was enthusiastic, the ability of these young students (5th and 6th graders) to grasp the concepts presented, even though two computers had to be shared by groups of up to 10 students, was remarkable. In addition several of the lessons were taken into each of the sixth grade classes of the school, again only two computers were available, each student was able to type only one item, but they were fascinated and very adept at acquiring computer skills.

The entire Workbook was used on a trial basis with computer classes at Columbus Junior High School and Canoga Park High School, including grades seven to twelve. At the secondary level, with the time frame of a regular public school semester in Los Angeles, it has been necessary to supplement this material with a textbook. The current choice for a secondary text is Programming in Basic, (Cummins, 1983).

Perhaps the most successful trial use of the material to date has been with adult in-service classes. Much of the workbook material was tried with the faculties of Columbus Junior High and Canoga Park High School.

Classroom Evaluation

Generally there are two purposes in evaluating materials, first to insure the effectiveness of the program, and secondly to locate areas where changes are needed in order to improve instruction and student retention.

The workbook material was evaluated by either orally questioning the students before beginning each instructional session, to determine their level of expertise on the subject of computers, or the use of a written pre-test, to determine attitude and knowledge. At the conclusion of each session the students were given written worksheets, quizzes, or practical tests to complete as post-testing instruments. The evaluation of student response to the presentation, and their performance on the post-testing instruments, was used to revise some of the materials. An example of this was the elimination of several items of technical information relating to the integrated circuit chips which provide sound and color capability within the machine, this material proved to be of little interest and unnecessary to the teaching of BASIC, therefore it was dropped from the workbook and the questions eliminated from the testing instruments.

In several instances product changes have required changes in the lessons. The introduction of the 800XL model is an example of an equipment change which has required several changes in the lessons to account for its differences.

Projected use of the Workbook

While the workbook is designed for an introductory course in ATARI BASIC, the individual lessons, taken separately would make effective springboard activities for the introduction of each included topic area. The "keyboard" exercises in lesson one, for example, are an effective way to begin any course using the ATARI, or ATARI software. The section dealing with Algebraic expressions in lesson two would be helpful to a math teacher in teaching students to translate formulas for use with the computer. Lessons, or activities from the lessons would also provide effective summaries for a unit of instruction. Art or music teachers could use the color or sound lesson to demonstrate and summarize principles in their fields. With proper structuring the activities could be used for the evaluation of instruction, use of exercises in mathematical functions and concepts, to test student knowledge in those areas, would be an example of how this could be accomplished.

This workbook material can be used in the upper grades of an elementary classroom, to develop or expand the understanding of programming and mathematical skills. Programming classes in both Junior and Senior High Schools, and adult in-service classes for teachers would both be effective areas for the use of this material. Additionally, the material used by an individual beginning the exploration of programming in BASIC is possible. Almost any instructional situation where a microcomputer, particularly ATARI hardware is featured, would provide an effective setting for the workbook, either as the primary instructional material or as a resource document.

Summary

Evaluation of the workbook through classroom testing has played an important part in the development of these instructional materials. At each level where it has been used, the material has proven effective. When a better means of putting an idea across was indicated, new ideas were encountered, or new equipment was introduced, the material was changed to reflect or incorporate the new products and ideas.

The use of individual lessons, activities, and the complete workbook were reviewed. Possible extended use of each, was discussed, but development of the work in this project has merely opened a door to a new areas of exploration. There is a need for advanced courses in BASIC, and courses in other languages like LOGO, and PASCAL. Development of instruction in programming to take advantage of the data or file management capabilities of the computer systems are needed. Finally the development of instruction in application programming where the student can learn to apply programming techniques for useful purposes, would prove beneficial.

BIBLIOGRAPHY

Albrecht, Bob, Finkel, LeRoy, Brown, J.L. ATARI BASIC. John Wiley & Sons Inc., New York, New York, 1979.

ATARI Corp. Invitation to Programming 1&2. ATARI ,Inc., Sunnyvale, Ca., 1981.

Bell, Norman. A computer Awareness Program for All Teachers and All Students. T.H.E. Journal Technical Horizons in Education. September, 1983, Pp. 138-141.

Bell, Hon. T. H. Effective use of Computers In Schools Requires Coordinated Developement. T.H.E. Journal Technical Horizons In Education. February, 1984, Pp. 80-83.

Bonner, Paul. Computer Programming: What's In It For You? Personal Computing, August, 1983, Pp. 128-137.

Is A Computer Hard To Learn? Consumer Reports, September, 1983, P. 488.

The Computer Primer. Electronic Learning, March/April, 1982, Pp.21-22.

Cummins, Jerry, Kuechmann, G. Programming in BASIC, Charles E. Merril Publishing Co. Columbus, Ohio, 1983.

DeVault, Vere, Harvey, J.G. Teacher Education and Curriculum Development in Computer Education. T.H.E. Journal Technical Horizons in Education, March, 1985, Pp. 83-86.

Frenkel, Cindy. How To Program Success Into Your Computer. Family Computing. September, 1983, Pp. 44-48.

Hopping, Lorraine. Eight Ways To Learn About Computers. Family Computing, October, 1983, Pp. 56-59.

Luehrmann, Arthur. Don't Feel Bad About Teaching Basic. Electronic Learning, September 1982, Pp. 23-24.

Martin, Diane, Heller, R. Presenting Computer Literacy for the B.C. Generation at the Smithsonian Institution. T.H.E. Journal Technological Horizons in Education, January 1984, Pp. 125-126.

People In computing. Personal Computing. December, 1983, Pp. 15-17.

Poole, Lon, McNiff, M. Cook, S. Your ATARI Computer, Osborne/McGraw-Hill, Berkeley, California, 1982.

Tandy Corp. Introduction to BASIC. Radio Shack. Tandy Corp., Fort Worth, Texas, 1979.

Walker, Decker. Reflections on the Potential And Limitations of Microcomputers. Phi Delta Kappan, October, 1983, Pp. 103-107.

APPENDIX A

INTRODUCTION TO BASIC PROGRAMMING: A STUDENT WORKBOOK FOR THE ATARI

INTRODUCTION TO BASIC PROGRAMMING: A STUDENT WORKBOOK FOR THE ATARI

This workbook will give an introduction to the microcomputer. Lesson one will familiarize the student with the ATARI keyboard and the screen editing procedures. Lesson two will introduce the concepts of BASIC programming statements and commands, and explore the treatment of mathematical functions on the computer. Lesson three will introduce program planning and the use of computer peripherals, including the disk drive and the printer. In lesson four, we will expand on the understanding of programming features. Lesson five will look at the BASIC concepts of branching and loops. The treatment of variables in BASIC is developed in lesson six. Lesson seven will explore functions programmed into the computer. In the concluding lesson, lesson eight, we will be introduced to the use of sound, color, and graphics in programming.

TO THE STUDENT:

As you work through these lessons you will be given very explicit instructions at some points and very general ones at other times. When instructions and programs are given it, is important that you follow them carefully until you understand the principles involved completely. You should then feel free to explore the area to see what you can do to expand your understanding and enjoyment of the computer and your programming skills.

TO THE TEACHER:

In this workbook there is sufficient material to teach a course in beginning BASIC. As you become more familiar with ATARI BASIC you will be able to add to the material and make it more interesting. The student should be encouraged to continue exploring each of the areas presented here. Those who complete this workbook should have a good start in understanding BASIC programming.

LESSON I:

Introduction to the Microcomputer

Purpose: The student will learn about the ATARI computer, the peripherals, and how to "hook-up" the equipment. There will be an exploratory experience with the computer keyboard, and keyboard editing.

NOTE: All answers will appear in parentheses.

Introduction is presented as a lecture by the instructor.

1. Introduction to the terminology.

CPU: Central Processing unit: the heart of the computer, ATARI uses the (6502 MICRO PROCESSOR), integrated circuit chip, the same as the APPLE.

Bit: The basic numerical unit for the (digital) computer, "ON" counts as one, "OFF" would represent zero.

Byte: The binary configuration of eight bits, it can represent a number up to (255) in decimal numerals.

K (kilobytes): One thousand bytes, actually 1024, used to figure computer memory, 64 K represents (64 THOUSAND) bytes of memory. It takes one byte to put one character on the screen.

ROM: Read only Memory, this is where all the (BUILT IN) functions of the computer reside, this memory cannot be erased, and is available each time the computer is turned on.

RAM: Random Access Memory, where programs and data are (STORED) in the computer, is erased each time the power to the computer is turned off.

Languages: Allows human communication with the (COMPUTER), BASIC, LOGO, PASCAL, FORTRAN, COBAL, C, AND FORTH are the most popular currently in use.

Cursor: Blinking square on the screen to show where next (CHARACTER TYPED) will appear.

Cartridges: ROM Programs, such as languages, word or data processors, and games; these plug into a (CARTRIDGE SLOT) for instant use.

2. Introduction to machine hardware (PERIPHERAL DEVICES), and hookup:

NOTE: hardware assembly will require demonstration by the instructor.

Video:

T.V.: ATARI uses (CHANNEL 2 or 3), attach adaptor, provided, to VHF connector, cable is also provided, may permit some outside interference with picture, but provides for full color and sound.

Monitor: May be (MONOCHROMATIC) or full color, with or without sound, no outside (INTERFERENCE), requires purchase of separate cable, may be more or less expensive depending on resolution and quality.

Power pack: (TRANSFORMERS), necessary for all components with integrated circuits, these decrease the voltage to prevent (ELECTRICAL ARCING) within the microscopic circuits of the components.

CAUTION: USE ONLY THE POWER PACK INTENDED FOR THE PARTICULAR COMPONENT.

Cassette Recorder: Slow but inexpensive (STORAGE DEVICE), has sequential rather than (RANDOM ACCESS), main advantage is sound capability, making narration of programs possible.

Disk Drive: Fast (HIGH VOLUME) storage device, uses random access, more dependable, but also more expensive.

Interface: (TRANSLATES) computer signals for use with printer or modem.

Printer: Several (TYPES), will be discussed later, it allows production of "hard" copy of material.

Modem: Permits communication between (COMPUTERS) and "bulletin board", information services through the use of the (TELEPHONE).

3. Turning on:

1) Video, This component needs time to (WARM UP).

2) Computer, All ATARIs can be turned on with or without the BASIC language being engaged.

400 - 800 Models: Will come on in (MEMO-PAD) mode

if the BASIC cartridge is not plugged in.

XL Models: Will come on in the (TEST MODE) if BASIC is disengaged by holding the (OPTION KEY) when turning on the machine, this is also necessary when loading a program which does not require BASIC.

800XL Test Mode: Allows testing various aspects of the CPU, such as: (MEMORY, KEYBOARD, and AUDIO/VISUAL)

3) Other Peripherals: Turning on, will be discussed as they are introduced.

4. The ATARI Keyboard:

Arrangement of (ALPHA-NUMERIC) keys (letters and Numbers), is the same as a standard typewriter. In addition there are many (specialized) keys on the keyboard.

CONTROL (CTRL) and SHIFT KEYS: Must be pushed down (BEFORE) and then (HELD DOWN) when using with another key, they do (NOTHING) by themselves.

RETURN KEY: Is used to (ENTER) information into the computer's (MEMORY). The computer will not recognize, acknowledge, or perform any activity until it has been (ENTER)ed. Pushing down the RETURN key tells the (COMPUTER) that you are finished. Then it interprets your line and gives an

(ERROR) message if it does not understand what to do with what you have written. The RETURN key must be used to ENTER BASIC (COMMANDS) or (STATEMENTS) into the computer.

BREAK KEY: Brings activity to a (HALT), or (STOPS) a program RUN. You can usually start again by typing (CONT) but not always, so be careful about hitting it by accident.

NOTE: The BREAK key may be used to advance the cursor to the next line on the screen without getting an ERROR message. When experimenting with the keyboard, use the BREAK key rather than the RETURN key to advance the cursor.

ESC (ESCAPE) KEY: Has various functions;

1) May be used in programming to (DELAY) action of functions which would normally take place immediately upon pushing a key. Example: Clearing the screen or Ringing (BUZZER).
2) It can be programmed to perform different functions depending upon the needs of the program or software used. It is pushed down one time before pushing another key. It is (NEVER) used while holding down another key. In (BASIC), it is always used in sequence to (SUSPEND) activities.

NOTE: This part of the exercise should be introduced by the instructor.

ATARI Keyboard Exercise:

1) Turn on Video, 2) turn on Computer, you see (READY) on the screen.

START: By typing each key alone to see what it does, record the results on the chart below. If nothing seems to happen when you push the key, then try typing several letters and see if anything has changed.

Notice what happens when a key is held down (CHARACTER REPEATS)

Notice what happens when the cursor gets to the end of a line without pushing the BREAK or RETURN key (JUMPS) to beginning of next line.

For this exercise use the (BREAK) key rather than the (RETURN) key to avoid getting (ERROR) messages.

NEXT: HOLD DOWN the SHIFT key and type the key again, record the results (hold key down if necessary to observe difference).

FINALLY: Adjust sound to hear a (BUZZER), when each key is depressed then, HOLD DOWN the CONTROL key and push each key a third time, record the results. Typing another character following, is sometimes necessary, to see the results of using a key with CONTROL. It may be necessary to move the cursor with the CONTROL (CTRL)+ one of the four (ARROW) keys to the middle of a line of characters to obtain the desired results.

Student Exercise 1:

Student Exercise 1:

KEYBOARD:

SYMBOLS	SHIFT + KEY	CONTROL + KEY
1/!	(EXCLAMATION)	(CURSOR WILL DISAPPEAR)
2/"	(QUOTES)	(BUZZER SOUNDS)
3/#	(NUMBER)	(ERROR MESSAGE)
4/$	($ DOLLAR)	(4 TO 9 NOTHING HAPPENS)
5/%	(PER CENT)	
6/&	(AMPERSAND AND)	
7/'	(APOSTROPHE)	
8/@	(AT)	
9/(	(OPEN PAREN)	
0/)	(CLOSE PAREN)	

NOTE: PARENTHESES: (ARE GROUPING SYMBOLS IN BASIC)

,/[	(BRACKET)	
./]	(BRACKET)	
/ ?	(QUESTION SHORT FOR PRINT)	
	(SLASH USED FOR DIVISION)	
ALPHABET	(UPPER CASE)	(GRAPHICS CHARACTERS)
CLEAR/<	(CLEARS SCREEN)	(ALSO CLEARS SCREEN)

NOTE: It will be necessary to have several lines of text on the screen for the next part of this exercise.

INSERT/>	(INSERTS LINE)	(INSERTS SPACE)
DELETE/ BACK SPACE	(DELETES LINE)	(DELETES SPACE)
TAB	(SETS STOPS)	(CLEARS STOPS)

NOTE: Default stops: (those set when machine is turned on), are 6 spaces for the first and 8 spaces for the remaining stops.

-/_/↑	(UNDERLINES)	(CURSOR UP ONE SPACE)
	(- HYPHEN OR MINUS SIGN)	
=/|/↓	(VERTICAL BAR)	(CURSOR DOWN ONE SPACE)
+/\/←	(REVERSE SLASH)	(CURSOR LEFT ONE SPACE)
*/^/→	(CARET)	(CURSOR RIGHT ONE SPACE)
	(ASTERISK USED FOR MULTIPLICATION, CARET USED FOR POWERS)	

NOTE: Cursor movement continues as long as ARROW keys are (HELD DOWN), when it reaches the edge of the screen it (JUMPS) to the opposite edge.

CAPS key (UPPER CASE)   (GRAPHICS CHARACTERS)

NOTE: Default for CAPS = CAPITALS when machine comes on, push key one time alone for lower case, once with CTRL key for graphics characters, once with shift key will return to regular (UPPER CASE) characters.

INVERSE KEY (BLACK/WHITE): Push (ONE TIME) to give inverse characters, (ONCE MORE) to return to normal.

Student Exercise 2:

KEYBOARD USE AND SCREEN EDITING

Use the BREAK KEY or CTRL + ARROW KEYS to move the cursor to the next line. DO NOT use the (RETURN) key which will give an ERROR message.

1. Make an alphabet of 2 of each letter. (AABBCCDDEEFFGG etc.)

Notice, it is in (UPPER) case, and that it wraps around (JUMPS) to the next line, use of the RETURN key is NOT necessary at the end of the line. Hit the BREAK key at the end of the entire alphabet, to move to the beginning of the next line.

2. Push the CAPS key ONE time, then make another double alphabet. (aabbccddeeffgg etc.)
   This one is in (LOWER) case.
   Now hold the SHIFT key and push the (CAPS) key to exit the lower case.
3. Push the INVERSE key and make a single alphabet, this time it will be in (INVERSE) video.
   Push the INVERSE key one more time to get back to normal.
4. Hold the CTRL key and push the CAPS key.
   Type another single alphabet. This one is really different! It is in a (GRAPHICS) mode
   Remember to push CAPS once to get back to the regular characters.
5. Hold the CTRL KEY + the four (ARROW) keys to move the cursor all around the screen.
   Move to the the upper left corner, the upper right, the lower right, the lower left. Now, aim straight off the top of the screen. What happened?
   The cursor (JUMPED) to the (BOTTOM) of the screen. Try going off the left or the right.
   Try the bottom, this is called (WRAP) around.

6. Move the cursor back up the screen, using the (CONTROL) + (UP) ARROW key, to the first alphabet you typed, the UPPER CASE alphabet. Change the second letter in each pair to an X. Use CTRL + the (LEFT) ARROW key to place the cursor, then just type over the old letter. The alphabet should now look like this: AXBXCXDXEXFX etc.
7. Move the cursor to the LOWER CASE alphabet. Eliminate 1 letter of every pair. Just typing over that letter will remove it, but there would be a (BLANK) left. So, use the CTRL key + DELETE/BACK SPACE to get rid of the spaces. Notice: the space was filled as the character was deleted. CTRL + DELETE BACK SPACE removes (ONE) space or character each time it is pushed. Your alphabet should read: abcdef etc.
8. CAREFUL NOW. Take the cursor to the line with the inverse video. SHIFT + DELETE BACK SPACE will (DELETE) an entire (LINE). This is a LOGICAL line, which may be up to (THREE) physical lines on the screen. A logical line ends whenever the (RETURN) key is pushed. Hold down SHIFT and push DELETE. The entire inverse alphabet (DISAPPEARED).
9. Move to the first letter of the lower case alphabet and try holding SHIFT + INSERT key. This will (INSERT) an entire line above

the cursor.

10. Move to the lower case alphabet again and add a second letter back into it. Use (CONTROL) + INSERT to do this. Place the cursor over the letter "b" to insert the space, and type a second letter "a". Use CTRL + (ARROW) to move over the "b". You should end up with: aabbccddeeff etc.
11. To CLEAR the screen. Use (SHIFT) + CLEAR, or(CONTROL) + CLEAR. They will both accomplish the same thing.
12. To use the keyboard graphics to draw a picture, (CONTROL) + (CAPS) will lock the computer into the graphics mode. Can you draw a wagon, a car, or something interesting?

    Have FUN!
13. How would you make inverse graphics? Push the (INVERSE) key then (CONTROL) + (CAPS).

NOTE: The instructor should review and demonstrate all covered material at this point.

QUIZ: (This is a practical quiz to be taken at the keyboard and observed by the instructor.)

ATARI KEYBOARD QUIZ

*** Do NOT write below this line. ***

One point for each item correctly completed.

You may proceed with the first four items.

____1) Type first name in Capital letters.

____2) Move cursor to next line type last name in lower case with only the first letter capitalized.

____3) Type your address, number and street, use upper and lower case letters.

____4) Sound the buzzer and wait for the instructor.

*** WAIT ***

____5) Move the cursor to the line where your first name appears, replace it with inverse video.

____6) Delete the line with your last name on it.

____7) Clear the screen.

_______SCORE

LESSON II:

Meeting ATARI BASIC

Purpose: This lesson will introduce students to computer commands in the BASIC language. They will learn to use direct commands to produce text and mathematical answers on the screen.

BASIC - Beginner's All-purpose Symbolic Instruction Code.

Student Exercise 1:

THE PRINT COMMAND

Look at each item below, what do you think will be displayed if each is typed into the computer? Write down your prediction before turning on the computer. Then, TYPE and ENTER each line EXACTLY as it appears. Use the (RETURN) key to enter each item. Record the computer's answer.

	TYPE	STUDENT ANSWER	COMPUTER ANSWER
1)	2+5	____________	(ERROR)
2)	PRINT 2+5	____________	(7)
3)	PRINT HELLO	____________	(0)
4)	PRINT "HELLO"	____________	(HELLO)
5)	PRINT "hello"	____________	(hello)
6)	print "OOPS"	____________	(ERROR)
7)	PRINT "this is ok"	____________	(this is ok)
8)	PRINT "25@'$*+.?>"	____________	(25@'$*+.?>)

Notice: When following PRINT (EVERYTHING) within the quotation marks will be displayed (EXACTLY) as it is TYPED. The computer does (NOT) perform any operation on anything within them.

Student Exercise 2:

MATH SYMBOLS AND ORDER OF FUNCTIONS

SYMBOLS: Addition (+), Subtraction (-),
Multiplication (*), Division (/)
Exponentiation (^).

Indicate what you think the answer for each of the following is. Then type each EXACTLY.

Remember: Press RETURN to ENTER the line when you have finished typing it.

	TYPE	STUDENT ANSWER	COMPUTER ANSWER
1)	PRINT "3-3"	____________	(3-3)
2)	PRINT "7*7	____________	(7*7)
3)	PRINT 4*4	____________	(16)
4)	PRINT 10/2+4-5	____________	(4)
5)	PRINT 8-2*3/2	____________	(5)
6)	PRINT 26+6/2-8	____________	(21)
7)	PRINT (3*6)/(3*3)	____________	(2)
8)	PRINT 3*6/3*3	____________	(18)
9)	PRINT 2^2	____________	(4)

Rules on the order of OPERATIONS: The computer follows the rules of (ALGEBRA) and performs (OPERATIONS) according to the following:

WORKING from (LEFT) to (RIGHT) through the problem, FIRST: it checks for (GROUPING SYMBOLS), performing any operations within (PARENTHESES).

SECOND: it performs any (EXPONENTIATION), raising each to its (POWER).

THIRD: it finds and performs, still from left to right, any (MULTIPLICATION) or (DIVISION).

LAST: left to right, it performs all (ADDITION) and (SUBTRACTION).

Try a few more difficult problems to be sure you've got it correct.

By the way, most microcomputers allow you to use a question mark "?" instead of having to type out the entire word PRINT.

	TYPE	STUDENT ANSWER	COMPUTER ANSWER
1)	? 25+(6/2-2)-8	____________	(18)
2)	? 200/(3+2)*2-60	____________	(20)
3)	? (6-2)*2*5	____________	(40)
4)	? 5+2*8*5	____________	(85)
5)	? (4/2+2)*3+(2+2)-12	____________	(4)
6)	? 3^2+6*4/2^2	____________	(15)

Student Exercise 3: (OPTIONAL)

ALGEBRAIC EXPRESSIONS

One of the things BASIC does well, is deal with ALGEBRAIC expressions.

These expressions must be re-written to conform to the principles of BASIC.

Compare, then evaluate the following:

ALGEBRAIC EXPRESSION	BASIC EXPRESSION	EVALUATION
1) $3^2+7\times5-4\div2$	?3^2+7*5-4/2	(42)
2) $6\times4+5^2\div1-7$	?6*4+5^2/1-7	(42)
TRY THIS ONE:		
3) $4\times3-1+2^3\div4$	?(4*3-1+2^3/4)	(13)
4) $3^2+\frac{6\times7-2}{3\times8-4}-5$	?3^2+(6*7-2)/(3*8-4)-5	(6)
5) $2+\frac{3(6+10)}{2(20+4)}$	?2+(3*(6+10))/(2*(20+4)	(3)

Student Exercise 4:

BASIC NUMBERS

What about numbers in BASIC? They don't always look the way you might expect them to. Some numbers must be typed into the computer differently than they would as a regular decimal number. (COMMAS) cannot be used in a BASIC number because they are used to separate DATA, or to display items in various print zones on the screen.

For example: ?123,456,789 will appear:

```
 (123       456       789).
```

For very large and very small numbers BASIC uses a variation of SCIENTIFIC NOTATION to display the numbers.

Compare the following:

	BASIC NOTATION	SCIENTIFIC NOTATION	E-NOTATION
1)	321000000000	3.21×10^{11}	3.21E+11
2)	0.00000321	3.21×10^{-6}	3.21E-06

You may have noticed that the power of ten in the number in scientific notation and the portion of the number in E-notation following the "E" are similar, leading you to conclude the "E" stands for (EXPONENT).

Try these: ?321000000000 AND ?0.00000321 are the results the same as the E-notation examples above? (YES)

What about these?

3) 0.000765 $(7.65X10^{-4})$ (7.65E-04)

Try some problems:

	PROBLEM	ANS: E-NOTATION	DECIMAL FORM
4)	5E+05+4E+02	(5.004E+05)	(500400)
5)	3.2E-02+3.2E+02	(3.20032E+02)	(320.032)
6)	45,000X.00023	(1.035E+01)	(10.35)

INTERESTING BITS OF INFORMATION:

This information is useful for screen editing, setting up program lines, of placement of screen graphics.

A Physical line is (38) characters or columns. It is referred to as a 40 column line, but the left margin is set in 2 spaces from the edge due to the curved corners cutting into the display on some T.V. screens. This can be eliminated and all 40 spaces used.

A Logical line is (114) characters normally, but it can be made to take 120. A bell rings at 107 to remind you to stop. All characters beyond the 114, are ignored by the computer. A logical line can be 1, 2, or 3 physical lines.

The shortest logical line is (1 CHARACTER). Pushing the (RETURN) key indicates the end of the logical line. Understanding the difference between a logical and a physical line will help in screen editing, since pushing CTRL + DELETE removes a (LOGICAL) line rather than a (PHYSICAL) line.

A Program line is treated as a logical line. The Computer acts as if it is 1 line. Many programs are written in only one line. Try typing this:

FOR I=1 TO 722: ?"A";:NEXT I:?"PHEW!" (REMEMBER PUSH RETURN) To check the amount of memory (ROM), type the following: PRINT FRE(0). A special print statement will tell us exactly how much memory (bytes) we have

left for use. Type PRINT FRE(0) and hit RETURN. How many bytes do you have? (?) Push the RESET key. Type PRINT FRE(0) again and write the number (?). Are they the same? (NO) Why not? (THE CHARACTERS USED IN THE SECOND LINE USED SOME BYTES OF MEMORY).

PROGRAMMED MODE or DEFERRED MODE:

Everything that we have done so far has been in DIRECT or (COMMAND) MODE. This means that the computer was instructed to do something as soon as the (RETURN) key was pressed. Programmed or Deferred mode provides the computer with a list of one or more operations. Then, when ready, the computer is told to do all of them in order one after another.

For this, we must use a (PROGRAM LINE) to tell the computer what order to do each item. To start the "execution" of the program, we "RUN" it, type (RUN). This is a COMMAND which tells the computer to do anything that it has been programmed to do.

To get rid of an old program when we want to do something different, we must type (NEW) or turn off the machine. Then the computer will empty its (RAM), ready for fresh information, or new directions.

Student Exercise 5:

It is time for your first program:

TYPE THE FOLLOWING: Don't forget to push the RETURN, <R> key at the end of each line to ENTER or EXECUTE it.

TYPE EXACTLY:

```
  10 PRINT "HOW" <R>
  20 PRINT "ARE" <R>
  30 PRINT "YOU?" <R>
```

CHECK for errors, if there are any move the cursor to that point and make the necessary corrections. Don't forget <R> after each line has been corrected.

NOW TYPE:

```
  RUN <R>
```

What do you see on the screen?

```
  (HOW)
  (ARE)
  (YOU?)
```

CLEAR the screen SHIFT + CLEAR. Is the program gone? (NO) The computer doesn't forget the program just because it has been RUN.

TYPE: RUN <R>, again. What happened this time? (SAME).

NOW TYPE: LIST <R>. What happened? (THE PROGRAM LINES WERE PUT BACK ON THE SCREEN IN THE ORDER OF THE LINE NUMBERS).

CLEAR the screen again.

TYPE: NEW <R>, RUN <R>, what happened? (NOTHING).

TYPE: LIST <R>, did anything happen? (NO).

Is the program really gone this time? (YES).

No two program lines can have the same (LINE NUMBER). When you type the same number a second time, what happens to the first one?

TYPE and RUN the following to find out.

```
  10 ? "LINE 10 ONCE" <R>
  10 ? "LINE 10 TWICE" <R>
  20 END <R>
  RUN <R>
```

The first line was eliminated automatically.

The LINE NUMBER determines the (ORDER) of the program execution.

TYPE: NEW <R>, DON'T FORGET THE <R>, NO MORE REMINDERS.

TYPE:

```
  30 ? "SINK"
  10 ? "SWIM"
  20 ? "OR"
  40 END
  RUN
```

NOW TYPE:

```
  5 ? "EITHER"
```

TYPE: RUN, what was the result (RUN) of this program?

```
  (EITHER)
  (SWIM)
  (OR)
  (SINK)
```

TYPE: LIST 20, what happened? (LINE 20 APPEARED ON THE SCREEN).

TYPE: LIST 20,40, what happened? (LINES 20, 30, AND 40 APPEARED).

TYPE: L., what happened?

(THE PROGRAM LINES WERE PUT BACK ON THE SCREEN IN THE ORDER OF THE LINE NUMBERS).

NOTE: L. is an acceptable abbreviation for LIST.

At times it is convenient to put things in your program to explain what a particular part of the program is doing, but you may not want it to show on the screen.

TRY THIS:

```
  2 WILL THIS APPEAR ON THE SCREEN?
```

WHOOPS!!!!!, An ERROR message!

TRY AGAIN:

```
  2 REM WILL THIS APPEAR ON THE SCREEN?
  RUN: did it appear? (NO).
```

REM stands for REMARK, abbreviated (.) and is used for documenting a program, or it explains what something in the program does.

REVIEW:

Definitions, commands, and statements.

BASIC:	(BEGINNER'S ALL-PURPOSE SYMBOLIC INSTRUCTION CODE).
RAM:	(RANDOM ACCESS MEMORY).
ROM:	(READ ONLY MEMORY).

STATEMENTS:		(TO BE EXECUTED LATER).
REM:	(.)	(REMARK, FOR DOCUMENTING PROGRAM).
PRINT:	(?)	(CAUSES MATERIAL TO BE DISPLAYED ON THE SCREEN).
COMMAND:		(EXECUTED AS SOON AS RETURN IS PUSHED).
PRINT:	(?)	(MAY BE EITHER COMMAND OR STATEMENT).
RUN:		(CAUSES PROGRAM TO BE EXECUTED).
LIST:	(L.)	(CAUSES PROGRAM LINES TO BE DISPLAYED ON SCREEN IN NUMERICAL ORDER).
NEW:		(ERASES RAM, OR DELETES PREVIOUS PROGRAM).

The BASIC language acts as an (INTERPRETER), it takes English and translates it so the computer's micro processor can understand it. Conversely, it takes what the (COMPUTER) says and lets us understand it.

LESSON III:

Steps in Planning, Saving, and Printing a Program

Purpose: This lesson will give the student an experience in planning a program through the use of flow charts. Then practice in the essential skills of operating a Disk Drive and formatting a Disk will be given. The use and operation of the Printer will be presented.

Program Development:

When an essay is written certain procedures are followed. First the assignment must be understood, then the plan of what is to be written must be put together. This is put on paper in the form of an outline, then the essay is written. It is edited to make sure there are no mistakes. Finally it is turned in. When a program is written essentially the same steps are followed.

FIRST: (UNDERSTAND THE SITUATION).
SECOND: (OUTLINE THE PROCEDURES).
THIRD: (WRITE THE PROGRAM).
FINALLY: (RUN THE PROGRAM AND "DEBUG" IT).

Student Exercise: 1

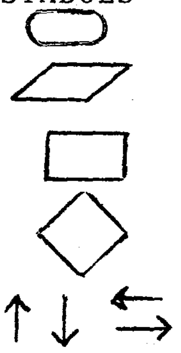
Suppose your assignment is to write a program to display the the numbers from one to ten, then compute and give their sum.

FIRST: The Situation: (THE NUMBERS FROM FROM ONE TO TEN

MUST APPEAR ON THE SCREEN. THEY MUST BE ADDED TOGETHER AND THEIR SUM DISPLAYED).

SECOND: OUTLINE THE PROCEDURES, Would a FLOW CHART help?

SYMBOLS FUNCTION FLOW CHART



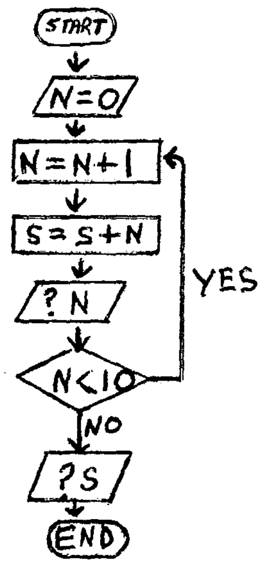
(START/STOP)

(INPUT/OUTPUT)

(OPERATIONS)

(DECISIONS)

(DIRECTION OF FLOW)



THIRD: WRITE THE PROGRAM,

```
10 LET NUMBER=0
20 LET NUMBER=NUMBER+1
30 LET SUM=SUM + NUMBER
40 PRINT NUMBER
50 IF NUMBER < 10 THEN 20
60 PRINT SUM
70 END
```

FOURTH: How would you test this program? (RUN AND DEBUG)

DEBUG: means (GET ALL THE ERRORS OUT OF THE PROGRAM).

TYPE, DEBUG, RUN, and LIST the program. Show the results to the instructor before continuing.

SCORE:_____

DISK DRIVE AND DOS

DISK DRIVE: A high speed, RAM (RANDOM ACCESS MEMORY) storage device, for data or programs, expensive, must be handled with care.

DISKETTES: (5 1/4) inch, floppy, single or double (SIDED), single or double density, soft (SECTORED). Most brands are OK.

STRUCTURE: Mylar, with an (oxide) coating which stores information as (MAGNETIC) signals. Protective jacket has a fabric lining to keep clean, a drive window, (READ/WRITE) window,and a (WRITE PROTECT) notch.

USE: To store large amounts of data or programs in a small area, very fast, convenient.

CARE: 1) keep in protective (ENVELOPE),

2) do NOT touch (READ/WRITE) window,

3) keep away from anything (MAGNETIC),

4) avoid extreme (TEMPERATURES),

5) don't (BEND) or fold,

6) keep away from food, drink, or dust.

FORMAT: New disk is (BLANK), and may be used by any computer with a 5 1/4 inch drive, but must be Formatted or (INITIALIZED). For the ATARI, this process divides the disk magnetically into 40 tracks, or (RINGS) like those on a record, but invisible. Each track has 18 sections, which gives (720) sectors.

TURNING ON MACHINES: following the correct procedure is very important here.

1) (VIDEO) and (DRIVE) may be turned on simultaneously. Actually the VIDEO may be turned on at any stage of the procedure, but needs time to warm up.

2) Insert disk with DOS (DISK OPERATING SYSTEM) in drive, (LABEL) or smooth side up.

3) Lock disk in place, turn locking lever from (horizontal) to (vertical).

4) Finally, turn on the computer.

5) When (READY) appears on the screen, TYPE (DOS <R>), the disk operating system menu should appear on the screen.

We are looking at: Disk Operating System II Version 2.05 Copyright 1980 ATARI.

There are other operating systems available.

MENU and USE OF ITEMS: allows the selection of any one item. Use only the (LETTER <R>).

A. Disk Directory-
B. RUN Cartridge-
C. Copy File-
D. Delete File(s)-
E. Rename File-
F. Lock File-
G. Unlock File-
H. Write DOS Files
I. Format Disk-
J. Duplicate Disk
K. Binary Save
L. Binary Load
M. Run at Address
N. Create Mem. Sav
O. Duplicate File

ITEM FUNCTIONS: Follow the prompts for each item to complete the function.

A. Provides a list of (FILES) or (PROGRAMS) on the disk.

B. Allows the use of any language, or software cartridge, returns the machine to (BASIC) on the XL models.

C. Makes it posible to duplicate a program at another location on the disk or a disk in another drive

D. Allows a file to be (REMOVED) from the disk.

E. Keeps the file but gives it a new (NAME).

F. (LOCKS) the file so it can't be changed or erased.

G. Unlocks the file so it can be replaced or (CHANGED).

H. Puts DOS and DUP (DISK UTILITY PROGRAM), on the disk so programs can be stored there.

I. Divides the disk into magnetic (SECTORS), for use with the ATARI.

J. Allows the copying of the contents of an (ENTIRE) disk onto another disk.

K, L, and M. Used with machine language programs.

N. Prevents the erasing of a program if it is necessary to reload (DOS). Must be done before the program is typed.

O. Permits the copying of a program or file on another disk.

Student Exercise 2:

FORMATTING A DISK:

1. Turn on the (VIDEO) and the (DISK DRIVE).
2. Insert an old Disk containing (DOS), lock it in place.
3. Turn on the (COMPUTER).
4. TYPE: (DOS) <R> to call or bring up the menu.
5. Insert a new (DISK) in drive.
6. TYPE: (A) <R><R> to check for anything on the disk, ERROR 144 means the disk hasn't been (FORMATTED), if you are sure that the disk is new, that is if you just opened the box, skip step 6.
7. TYPE: (H) <R>, to Format the Disk, watch the screen for instructions.

8. Answer the prompts, or questions on the screen:

   What drive...? TYPE: (1) <R>.

If there is more than one drive, then type the number of the drive which contains the (NEW) disk. Be very careful that you don't format the OLD disk.

Type "Y" to Format Disk? Type (Y) <R>.

WARNING: FORMATTING WILL ERASE ANYTHING THAT MAY ALREADY BE ON THE DISK. Whenever you see Type "Y" to ..., STOP, think, and be sure, before continuing.

Listen to formatting if drive is loud enough. Prompt will appear when disk is Formatted.

9. Type: (A) <R><R> to check if formatting was successful. You should see (707) Free Sectors.
10. Type: (H) <R> to Write DOS Files onto Formatted disk, see menu. Follow prompts: Drive..., (1) <R>. Then (Y) <R> to Write Files.
11. When complete, prompt will appear. TYPE: (A) <R><R> again to check for DOS and DUP Files, prompt should read (626) free sectors.

If you read 626 free sectors, congratulations: you have successfully formatted a disk.

Wait for the instructor before going on.

SCORE______

Student Exercise 3:

SAVING A PROGRAM:

It's time to SAVE your first program:

HELP how do we get back to BASIC? TYPE: (B <R>).

TYPE: the following program. Don't forget <R>. Give your own information for items in lower case.

```
10 ? "HELLO
20 ? "MY NAME IS (your name)
30 ? "GOODBYE
40 GOTO 10
50 END
```

LIST, TYPE: (L.), correct any errors. Did you include your name?

NOTICE: the (CLOSING QUOTES) have been included in the listing.

TYPE, and RUN: How do we STOP this? (BREAK) key.

Let's save the program:

TYPE: SAVE "D:(name) <R>

Did the busy light go on?

Check the directory:

TYPE: DOS <R>, when the DOS (MENU) comes up, <A>,<R><R>.

Is your "name" on the directory?

If so you have successfully saved your first program.

Remove Disk from drive, Turn off computer, Turn off drive.

RUNNING YOUR PROGRAM:

Turn on (DRIVE). When (BUSY LIGHT) goes off, (INSERT) disk. Turn on (COMPUTER), when READY, call up (Type) (DOS). Call Disk Directory: (A) <R><R>. Identify correct name of File or (PROGRAM).

TYPE: RUN "D:(name), just exactly the way you saved the program. Did your program RUN? GREAT! TRY, changing a line then save the program again. Back to BASIC.

TYPE: LOAD"D:(name), TYPE: L., is your program there?

LOCK: your file, check the directory, what's different?
(AN ASTERISK).

DELETE: your file, what happened? (ERROR).

UNLOCK: it and try DELETING again. Check the directory.
Did it work this time?

REMOVE disk before turning off hardware.

FILENAMES and EXTENDERS:

A Filename may contain up to (8) letters or (NUMBERS), must start with a (LETTER), contain no (SYMBOLS) or (SPACES). An (EXTENDER) of up to (3) letters or numbers may be added, to the end of a file name. Between the regular filename and the Extender, there must be a (PERIOD). Extenders frequently indicate the (LANGUAGE) used to execute the program, or a special software package like (AW, FOR ATARI WRITER).

REMEMBER, 8 letters, a ".", and an extender (if used), NO spaces or symbols.

USING A PRINTER:

TYPES OF PRINTERS:

DOT MATRIX: Most have 9X9 array of pins which impact the paper through a ribbon. They are fast, durable, and will do carbons, dittos, and graphics, but are

(NOISY). The cost is moderate.

LETTER QUALITY: Impacts through a (DAISY WHEEL), has very high quality, but is expensive, and slow and loud.

INK JET: Shoots ink from a (CARTRIDGE) onto the paper. Very quiet, inexpensive, and durable. They are messy, and don't give an impression for carbons or dittos.

THERMAL: Requires heat sensitive (PAPER), is very quiet, clean, inexpensive to buy, but expensive to operate, due to the cost of the paper, There is no impact on the paper.

FEATURES to look for:

FRICTION ROLLER: Allows the use of stationary, dittos, rolls and single sheets (OF PAPER).

TRACTOR DRIVE: Provides for continuous (FEED) of folded paper or forms, and much more precise positioning of them.

Having BOTH friction and tractor, is the best situation, but a friction roller is preferable in most cases if a choice must be made.

INTERFACE: (TRANSLATES) computer language for the printer and other peripherals.

CABLES: Must have the proper wiring and end (CONNECTORS), may have interface built in.

CONTROLS: This information is specifically for the GEMENI 10X but applies to many Dot Matrix Printers.

HOOK UP: Plugs into regular outlet, doesn't require a transformer. Usually routed through the back of the disk drive to the computer.

POWER SWITCH: Located on the right side, may be turned on at any time prior to sending print command from computer. Certain programming may require the printer to be turned on at a specific time to receive special formatting instructions.

ON LINE: Sets printer to receive print commands.

F.F.: (FORM FEED), advances the paper to the setting required for the next printing operation.

L.F.: (LINE FEED), moves the paper up one space each time it is pushed, or continuously when held down.

NOTE: The ON LINE button must be pushed before FF or LF will function.

DIP SWITCHES: Control the type of print, the size of spaces, the form feed, and other aspects of the print on the page. It is necessary to check the printer manual for all the variations.

Student Exercise 4:

TURNING ON:

1) Turn on the video, disk drive, and the computer
2) When ready turn on the printer.
3) Check positioning of paper.

WARNING: Do NOT move the paper roller with the power turned on, damage to the printer drive mechanism may result.

4) Check the POWER, ON LINE, READY and

PAPER OUT lights.

BASIC COMMANDS: which will be used with the printer.

LINE PRINT (LP.): Will replace ? in the program to send the line to the printer rather than the screen. If both screen display and "hard copy" are desired, two Print lines may be necessary.

L."P: (LIST device, PRINTER), sends a listing of the program to the printer.

TYPE: and SAVE the following BASIC program:

```
10 LP. "(your name)
20 LP. "(today's date)
30 FOR N=1TO5
40 LP. "TEST",N
50 NEXT N
60 LP.:LP.
70 LP. "THAT'S ALL FOLKS
80 LP.:LP.:
90 END
```

To save this program:

TYPE: SAVE "D:PRINTEST.(put your initials here for the extender so we can tell who it belongs to, but remember no spaces.

Take your disk to printer station:

TYPE: LOAD "D:PRINTEST.(initials)

TYPE: L. (to list and check the program on the screen.)

CHECK: Is the printer ON LINE?

RUN

TYPE: L."P (to list program on the paper, "Hard copy")

Show "print out" to instructor for evaluation.

SCORE_____

LESSON IV:

Additional Programming Features

Purpose: In this lesson we will look at some new commands: END and STOP, and some programming techniques which will make many routines easier. Messages from the computer will be explored. The INPUT, READ/DATA, and RESTORE statements will be looked at as they effect putting information into the computer.

More Commands:

END:

Terminates execution of the program whenever (COMMAND) is encountered. With most microcomputers, END is not necessary in BASIC, however, it is needed on some larger computers, with some other programming languages, and can serve a purpose here, as will be discussed later.

STOP:

Functions the same as the END command, but includes the PROMPT: (STOPPED AT LINE _?_). It serves a useful purpose when debugging longer programs, and prevents certain routines from being exicuted unintentionally.

Student Exercise 1:

ENTER AND RUN the following:

```
10 ?"HELLO
20 ?"HOW ARE
30 END
40 ?"YOU?
RUN
```

What is the output?

(HELLO)

(HOW ARE)

DELETE: line 30, TYPE: 30 <R>.

LIST: is line 30 gone?

RUN: what is the output this time?

ENTER AND RUN:

```
10 ?"THE SUM OF 3+4 IS ";
20 ?3+4
30 STOP
40 ?"2 TIMES 3 =
50 STOP
60 ?2*3
70 END
```

Give the output.

(THE SUM OF 3+4 IS 7)

STOPPED AT LINE (30)

TYPE: CONT, the out put is:

(2 TIMES 3 =)

(STOPPED AT LINE 50)

TYPE: CONT: What is the function of the CONT command? It causes the program to continue after the (STOP).

Student Exercise 2:

ENTER and DEBUG the following program:

```
PROGRAM                CORRECTIONS               OUTPUT

10 RAM ODD INTEGERS    (10 REM ODD INTEGERS       (1)
1-11 AND THEIR SUM     (1-11 AND THEIR SUM)       (3)
20 N=1                 (OK)                       (5)
30 S=5                 (S=0)                      (7)
40 print n             (PRINT N)                  (9)
50 S=S-N               (S=S=N)                    (11)
```

```
 60 N=N+3             (N=N+2)                   (__)
 70 IF N>11 THEN  10  (IF N<11 THEN  40)        (36)
 80 ?"___             (OK)
 90 ? RUN             (? S)
 RUN
```

Did you get the correct output?

Check with instructor.

Score____

BASIC Message Responses

Student Exercise 3:

ERRORS: You've already seen some ERROR messages. These appeared because the computer didn't understand what you typed.

1. COMPUTER DETECTED ERRORS:

   There are 2 types of computer detected errors: SYNTAX and RUN-TIME errors.

   A. SYNTAX ERRORS: are those BASIC language errors the computer detects when the RETURN key is pushed.

      Examples:

      1. Something left out or incorrect: (;/: ./, NO QUOTES, SINGLE (, NO LINE NUMBER)
      2. Additional characters present: (GOTO, 20; ":" AT END OF LINE)
      3. Command name is incorrect for ATARI BASIC: (HOME, FLASH, SET, etc.)
      4. Indicated character or word has no meaning to ATARI BASIC within the context of the statement being examined: (PINT, RUM, PRINTY), usually

a spelling error.

B. RUN-TIME ERRORS: Errors in the program (INSTRUCTIONS).

Detected when attempt is made to execute (STATEMENT).

Examples:

1. Variable out of (RANGE) or off the screen
2. Missing a necessary part of a command: FOR without (NEXT), THEN no (IF), missing (DATA).
3. INCORRECT NESTING OF (FOR/NEXT) loops.
4. Division by (0).
5. GOTO or (GOSUB) with non-existent line numbers.
6. GOSUB with no (RETURN), or VISA VERSA.

2. LOGICAL ERRORS: Definition doesn't violate rules, but gives incorrect (RESULTS).

Example: using the formula for the area of a triangle when the area of a rectangle was intended, putting the wrong DATA into the program, or putting the wrong operation symbol like / instead of *.

INFORMATIVE MESSAGES:

Not all messages from the computer are "ERROR" messages, some are programmed merely to give information. These non-ERROR messages are sometimes called (PROMPTS)

1. READY which indicates:

a. Task is (COMPLETED).

b. (BREAK) Key was pressed pressed.

c. (RESET) Key was pressed.

2. STOPPED AT LINE _?_ which indicates that:

a. (BREAK KEY PRESSED).

b. (RUN TIME ERROR).

c. (STOP in program).

INPUT statements:

There are several ways to get information or DATA into the computer. So far we have looked at the "LET" statement as a means to assign values to variables. The next statement we will work with allows the assignment of values to variables from outside the program. When the computer executes an INPUT statement, it (WAITS) for an entry from the keyboard. Until the computer gets the (INPUT) it requires, nothing else will happen. Compare the following:

```
   LET                  INPUT
 10 LET N=5           10 INPUT N,M
 20 LET M=10          RUN

                      ?5,10
```

In its simplest form, an INPUT statement begins with the word INPUT and is followed by a variable name. Data entered from the keyboard is assigned to the variable, after the program execution has started.

Student Exercise 4:

TYPE and RUN the following program and when the (?) appears on the screen, type some different numbers.

```
10 INPUT A
20 PRINT A
40 GOTO 10
RUN
```

Upon executing an INPUT statement, the computer displays a question mark, then waits for your entry. The program above displays each (NUMBER) as you ENTER it. REMEMBER to press the <RETURN> key to (COMPLETE) your entry. Why do you think this program displays every number twice?

HOW DO YOU STOP THIS PROGRAM? (BREAK KEY)

ADD line 30 below, RUN again, try various numbers.

```
 30 IF A=0 THEN END :REM End program if 0 is entered
```

NOW try 0, what happened? (PROGRAM ENDS).

The INPUT statement can accept more than one value at a time. The variables are separated with (COMMAS). When an INPUT statement is executed, there are two ways to enter the DATA, but you must respond with a separate value for each variable.

Change lines 10 and 20 to those below.

```
10 INPUT A, B, C
20 ? A, B, C
RUN
```

TRY entering a number each time the "?" appears. Be sure to press the RETURN key after each value.

Now try entering three numbers with commas between them. What happened? Try two numbers and then RETURN. What happened?

You should see that if more than one number is called for, they can be entered together or separately, but the computer will not continue until they have all been (ENTERED).

REMEMBER: in large numbers, use 1000000 not 1,000,000.

READ/DATA:

Let's look at one more way to enter data into the computer, the READ and DATA commands MUST be used together. The DATA statement is always in the (DEFERRED) mode in a program, it is usually located at the (BEGINNING) or (END) of the program, rather than somewhere in the middle, since it is not really a part of the program itself.

Look at this program and try to determine the output.

ANTICIPATED OUTPUT: ____________________________________

```
  10 DATA 2,4,6,8
  20 READ M,N,O,P
  30 ? M
  40 ? N
  50 ? O
  60 ? P
  RUN
```

How did your prediction turn out?

Look at the program below and note how the READ/DATA statement pair is used. What will be the outcome here?

____

```
  10 DATA 2,4,6,8,10,99
  20 READ A
  30 IF A=99 THEN 100
  40 ? A
  50 GOTO 20
  100 END
```

READ/DATA must follow these rules:

1. (DATA) must be assigned by a (READ) statement.
2. DATA must be separated by (COMMAS).
3. DATA must be read from (LEFT) to (RIGHT).
4. DATA may be located anywhere in the program
5. The number of READ variables and the number of DATA (ELEMENTS) must always be (THE SAME). If there are more "READS" than there are "DATA" elements you will get an (OUT OF DATA) ERROR.
6. The DATA statement (CAN CONTAIN) both numbers and words.
7. The computer will (NOT) evaluate (ARITHMETIC OPERATIONS) or (FUNCTIONS). What you see is what you get.

What will be the out put of this program? (1 2 3 4 5 6 ERROR)

```
10 READ A
20 ? A
30 GOTO 10
40 DATA 1,2,3
50 DATA 4,5,6
RUN
```

TYPE, these lines:

```
30 GOTO 60
60 IF A = 6 THEN RESTORE
70 GOTO 10
RUN
```

What did RESTORE do? Allowed the (DATA) to be used over and over.

RESTORE:

The ATARI contains an internal pointer which keeps track of the DATA item to be read next. The (RESTORE) statement (RESETS) that pointer to the first (DATA) item in the program. This permits the reuse of the same DATA over again. RESTORE may also be used with a line number in which case it will reset to the first item on that line.

Example: Replace line 60 in the program above, with:

```
RESTORE 20
```

What happened?

Experiment with the RESTORE statement to see what effects you can achieve.

LESSON V:

Branching and Loops

Purpose: In This lesson we will look at two types of branching, where the program goes out of the normal sequence of line numbers, as it is executed. There are other branching techniques called loops which will also be explored here.

Branching:

Each time there is a decision to be made in a program, we think of it as a branch. The computer must analyze the information or data and decide which choice to make. There are essentially two types of branches: an unconditional branch, in which the computer must go in the direction indicated, and a conditional branch, where certain "conditions" must be met before the computer will execute a branch from the program.

The Unconditional Branch:

Student Exercise 1:

Examine the program below. What is the function of each line?

What is in the RUN? Complete the chart, indicating what will take place in the computer memory and on the screen.

```
PROGRAM            LINE FUNCTION     MEMORY - SCREEN
 10 N=0            (INITIALIZE)     (N=0    -       )
 20 N=N+1          (ADD ONE)        (N=1    -    1  )
 30 ? N            (DISPLAY #)      (N=2    -    2  )
 40 GOTO 20        (BRANCH)         (N=3    -    3  )
 50 ?"THE END                       (N=4    -    4 ETC.)
```

Will line 50 ever be executed? (NO).

KEEP the program in your machine. We will work with it again.

Notice we have used statements including "GOTO" in several of our practice programs.

The GOTO statement: is known as an (UNCONDITIONAL BRANCH), because each time the computer reaches it in the program it must transfer control to the (LINE INDICATED).

When the computer reaches line 40 in the program above, line (20) is the next to be executed.

GOSUB is also an unconditional branch, but we will cover it in more detail later.

The CONDITIONAL Branch:

CHANGE line 40 from the program listed above, to read as follows: ENTER and RUN.

```
  40 IF N<10 THEN 20
```

What is the output of the program now?

(1 2 3 4 5 6 7 8 9 10 THE END).

Was line 50 executed? What is the condition set up in line 40, for branching to occur? (THE NUMBER MUST BE LESS THAN TEN). What happens if this condition is not met? (THE PROGRAM "FALLS THROUGH" TO THE NEXT LINE).

THE IF/THEN STATEMENT: these program statements provide for a (CONDITIONAL LOGIC) sequence. IF a given condition or set of conditions exist, the computer performs the instruction present in the THEN command. If the conditions are not met, the computer ignores the THEN portion of the statement and goes to, (FALLS THROUGH TO) the next program line.

LOGICAL OPERATORS used in determining (CONDITIONS) are:

=	means (EQUAL)
<>	means (NOT EQUAL)
>	means (GREATER THAN)
<	means (LESS THAN)
<=	means (LESS THAN OR EQUAL TO)
>=	means (GREATER THAN OR EQUAL TO)

KEYWORDS: AND, NOT, OR are also operators.

Let's go back and make another change in our program, then predict the results:

What will the output be? (2 4 6 8 ALL DONE).

```
20 N=N+2
50 PRINT "ALL DONE"
```

Was the output as you predicted?

Complete the statements for this program which counts to 30 by 3's.

Indicate what each line is doing.

```
LINE                          FUNCTION
10 N= (0)                   (ASSIGNS 0 TO N)
20 N= (N+3)                 (ADDS 3 TO N)
30 ?  (N)                   (DISPLAYS THE VALUE OF N)
40 IF (N<30 THEN 20)        (LOOPS UNTIL N = 30)
50 PRINT "THE END"           (DISPLAYS "THE END)
```

TYPE, ENTER, DEBUG and RUN your program. Did it RUN correctly?

The following program illustrates the use of multiple IF/THEN statements. The computer is told to compare and decide.

REMEMBER, the message in an IF/THEN PRINT statement will be printed ONLY if the comparison is true.

TYPE: NEW, then ENTER and RUN this program

```
10 REM ** DETERMINE IF X IS POSITIVE, NEGATIVE OR ZERO**
20 ? "I WILL ASK YOU TO ENTER A NUMBER, THEN I "
30 ? "WILL TELL YOU WHETHER YOUR NUMBER IS "
40 ? "POSITIVE, NEGATIVE, OR ZERO"
50 ?
60 ? "WHAT IS YOUR NUMBER";
70 INPUT X
80 IF X>0 THEN ? "YOUR NUMBER IS POSITIVE"
90 IF X<0 THEN ? "YOUR NUMBER IS NEGATIVE"
100 IF X=0 THEN ? "YOUR NUMBER IS ZERO"
110 GOTO 50
```

Try several numbers, then answer the following questions.

1) What is the condition in line 80? (X MUST BE GREATER THAN 0)

2) What is the condition in line 90? (X MUST BE LESS THAN 0)

3) What is the condition in line 100? (X MUST EQUAL 0)

4) What does line 110 do and why doesn't it use line 20 instead?

   (THERE IS NO NEED TO GIVE THE INSTRUCTIONS OVER AGAIN).

5) If your program is still going on, what do you have to do to stop it? (use the BREAK KEY).

FOR/NEXT LOOPS:

REVIEW:

The simplest way of causing a program to repeat is to use a GOTO which sends you back to an earlier line.

```
10 PRINT "OVER AND OVER"
20 GOTO 10
RUN
```

If you want to loop only a certain number of times, you can use a variable which increases each time you go through the loop. When the variable reaches a certain value, you can exit using an IF X=n...THEN GOTO command. Note it is not necessary to use the word GOTO, though the computer will accept it here.

```
  10 ? "OVER AND OVER
  20 X=X+1
  30 IF X=10 THEN 50
  40 GOTO 10
  50 ? "FINISHED!"
```

In the program above, which begins with line 10, progresses through lines 20,30, and 40 with some activity, and ends with line 50, the following will have taken place:

1) The Variable X starts at 0, the default value
2) Next it is incremented by one each time through the loop
3) It continues until X reaches 10
4) Then the condition of the loop is no longer met
5) It then "falls through" to line 50 and ENDs.

A FOR/NEXT command is a third way to accomplish this same kind of thing. The FOR/NEXT counting loop repeats a program segment by executing statements contained within the loop a given number of times.

The FOR....NEXT command is actually two commands. The (FOR) command is placed at the (BEGINNING) of a section of the program that you wish to repeat.

It appears as follows:

```
  10 FOR X = 1 TO 10
```

The "X" can be any arithmetic variable. The first number "1" sets the initial value of this variable. The second number "10" sets the upper limit of the variable. In this loop the increment, or the adding to the number each time through the loop is automatic.

The (NEXT) command is found at the (END) of the section to be repeated. It appears as follows:

```
  50 NEXT X
```

The NEXT statement in line 50 tells the computer to return to the FOR statement in line 10 and proceed back down to line 50 again. Each time the computer encounters this program line, X increments it by 1 and returns to (REPEAT THE LOOP AGAIN) until the program lines have been executed the number of times specified by the upper limit of the (FOR) statement.

TYPE and RUN:

```
  10 FOR NUMBER = 1 TO 99
  20 ? NUMBER
  30 NEXT NUMBER
  40 ? "THE END"
```

The computer has counted by "ones" in executing this program. We can tell the computer to use something other than one, by typing the STEP N function. (where N = the number we wish to count by.)

Try changing line 10, RUNning it, and then write your results.

TRY THESE:

```
  10 NUMBER = 1 TO 99 STEP 2
```

RESULTS: (1 3 5 7...99)

```
  10 FOR NUMBER = 1 TO 99 STEP .5
```

RESULTS: (1 1.5 2 2.5 3 3.5...99)

```
  10 FOR NUMBER = 100 TO 1 STEP -2
```

RESULTS: (100 98 96 94...2)

Note: when the STEP is negative the upper and lower limits must be reversed.

You can use the FOR/NEXT loop for many things: (Try each and record your results)

A. To delay the computer while it counts up to the limit you

have set. This can be used to hold something on the screen for a specific amount of time.

Add this line and RUN the program again.

```
  25 FOR D=1 TO 1000:NEXT D
```

RESULTS: (WAITS UNTIL COUNT HAS FINISHED BEFORE GOING ON)

B. To play a musical scale.

Note: adjust sound volume before RUNning this program.

```
  10 DELAY = 100
  20 FOR P=1 TO 255
  30 SOUND 0,P,10,10
  40 FOR D=1 TO DELAY:NEXT D
NOTE: This delay loop slows down the sound.
  50 NEXT P
  60 END
```

Note:END will turn off SOUND

TRY: changing the value of DELAY in line 10 and indicate your results. (WILL BE LONGER OR SHORTER DEPENDING ON VALUES USED)

C. To draw a pattern.

```
  10 GR. 7
  20 COLOR 3
  30 FOR X=1 TO 80 STEP 5
  40 PLOT 2*X,X
  50 NEXT X
```

FOR/NEXT LOOPS are so common in programs that they are frequently (NESTED) one inside the other like a set of mixing bowls. There can be any number of statements between the FOR and the NEXT. Frequently there are tens, or even hundreds of statements. And within these tens or hundreds of statements, additional loops may occur. Since ATARI BASIC allows (128 DIFFERENT) variables, you can have at most 128 levels of FOR/NEXT NESTING.

Just remember that each (NESTED LOOP) must be completely inside the next outer loop.

NOTE: You may not close an outer loop before you close the inner loop.

Try this example:

```
  10 GRAPHICS 7:COLOR 3
  20 FOR X=50 TO 100 STEP 2
  30 FOR Y=50 TO 20 STEP -2
  40 PLOT X,Y
  50 NEXT Y
  60 NEXT X
```

REMEMBER, we can put several commands on one line, but we must still close the loops in the proper (ORDER).

TYPE and RUN.

```
  10 FOR OVER = 1 TO 3
  20 FOR FIRST=1 TO 255 STEP 50
  30 FOR SECOND=1 TO P1
  40 SOUND 0,FIRST,10,10
  50 SOUND 1,SECOND,10,10
  60 NEXT SECOND:NEXT FIRST:NEXT OVER
  70 END
```

What do you hear?

SUBROUTINES

The GOSUB statement branches in the same way as a GOTO, but in addition it remembers the (LINE) number from which it branched. It will then (RETURN) to the line following. In computer jargon, we say GOSUB calls a (SUBROUTINE).

You MUST end the subroutine with a (RETURN) statement. It causes a branch back to the statement which follows the original GOSUB statement.

TYPE and RUN:

```
  10 REM SUBROUTINE PROGRAM
  20 ? "THE 1ST TIME:"
  30 GOSUB 1000
  40 ? "2ND TIME:"
  50 GOSUB 1000
  60 ? "3RD TIME:"
  70 GOSUB 1000
  80 STOP
  1000 ?
  1010 ? 1,2,3,4,5
  1020 ?
  1030 RETURN
```

Take line 80 out of the program. What happened?

(SUBROUTINE WAS USED AN EXTRA TIME, WHEN <RETURN> WAS REACHED AN ERROR OCCURRED).

This next program has 3 subroutines. Can you tell what the results will be BEFORE you RUN it? RESULTS:

```
   (++++++++++)
   (----------)
   (**********)
   (----------)
   (++++++++++)
```

```
  10 REM PROGRAM WITH SUBROUTINES
  20 GOSUB 100
  30 GOSUB 200
  40 GOSUB 300
  50 GOSUB 200
  60 GOSUB 100
  70 END
  100 ? :? "++++++++++":RETURN
  200 ? :? "----------":RETURN
  300 ? :? "**********":RETURN
```

GOSUBs in ATARI BASIC can also use variables instead of numbers. Change the program as follows:

```
  10 REM PROGRAM WITH SUBROUTINES USING VARIABLES
  15 LET PLUS=100:LET MINUS=200:LET TIMES=300
  20 GOSUB TIMES
  30 GOSUB MINUS
  40 GOSUB PLUS
  50 GOSUB MINUS
  60 GOSUB TIMES
  70 END
  100 ? :? "++++++++++":RETURN
  200 ? :? "----------":RETURN
  300 ? :? "**********":RETURN
```

Get the computer to display the name of each subroutine. Type:100 ? :? "PLUS";"++++++++++":RETURN and change the other two accordingly.

ON GOTO and ON GOSUB are used to send the computer to different lines as the program is run. The difference between them is that (ON GOSUB) will RETURN to the line following the (ON GO SUB)statement, while the (ON GOTO) will continue from the line to which it was sent. Try the following example of ON GOTO.

```
  10 ?"TYPE IN 1 FOR A HOT MESSAGE, 2 FOR A COOL MESSAGE,
     AND 3 FOR A COLD MESSAGE";
  20 INPUT X
  30 ON X GOTO 60,90,110
  30 ON X GOTO 60,90,110
  40 ?
  50 GOTO 10
  60 ?"IT'S 100 DEGREES"
  70 FOR A=1 TO 500:NEXT A
  80 GOTO 10
  90 ?"ICE CREAM CONE"
  100 GOTO 70
  110 ?"A BLIZZARD"
  120 GOTO 70
  130 END
```

What is the purpose of line 50? (IT SENDS THE PROGRAM BACK TO START OVER IF A WRONG CHARACTER IS PRESSED). What is the name given to a program line which does this? (ERROR TRAP)

What is the purpose of lines 70 and 80? (70 IS A DELAY LOOP, TO KEEP RESPONSE ON SCREEN FOR SEVERAL SECONDS, 80 SENDS COMPUTER BACK FOR NEW RESPONSES).

Try to convert this program to an ON GOSUB, removing any extra lines. Lines 10, 20, 40, and 50 are ok, change the GOTO in line 30 to (GOSUB)

```
  40 (FOR A=1 TO 500: NEXT A)
  50 (GOTO 20)
  60 (?"IT'S 100 DEGREES":RETURN)
  90 (?"ICE CREAM CONE":RETURN)
  110 (?"A BLIZZARD":RETURN)
  120 (END)
```

One use of the ON GOSUB and ON GOTO subroutines is to eliminate repetition of (RESPONSES). Many times a program will go to the same response every time the computer receives a certain input. For instance, each time a student gets a math problem or spelling word correct the computer prints "GREAT WORK". By using ON GOTO or ON GOSUB with a (RANDOM FUNCTION) generating a 1, 2, or 3, the computer can print "RIGHT ON", or "YOU GOT IT. KEEP IT UP", ETC., in response to correct answers.

LESSON VI:

VARIABLES

Purpose: We have been using letters and words to represent values through out these lessons, it is time to see what the functions and limitations of these VARIABLES are. We will see that it is necessary to DIMENSION, to save memory space for word or string variables. Then we will work with subscripted variables to create arrays and matrices.

Variables:

To the computer a variable is like the street address is to a house. The first time it is used in a program a variable is assigned a location in memory where a value can be stored.

1. Variables identify (MEMORY LOCATIONS) in which the computer stores specific (ALPHABETIC) or (NUMERIC) characters, the result of a computation or an assigned value. Assignment statements let you assign values to variables. Some examples include:

```
10 LET X=3        10 W=56
10 A$="ANYONE"  10 NAME$="SUSAN"
```

2. Variable types determine the kind of labels required. Numeric variables reference numbers ONLY. String, (WORD) variables store any assigned sequence of (ALPHA) or (NUMERIC) characters or their combination.

   STRINGS can include any combination of letters,

words, or numbers up to (255) characters.
A$="ARF", GIRL$="5", M$="MUSIC", etc. are
all possible. Just remember to use (QUOTATION)
marks to indicate the entire string to the computer.
Each item of information is given a name which will
be used throughout the program to refer to that
particular item. This name is an identifier or label.
As each item is given a label, (A VARIABLE NAME)
it is stored by the computer in a memory box.

We say A=15, B$="HELLO", HEADS=1, or SIDE=X, see below:

```
    MEMORY BOXES
    A=15          B$="HELLO"    HEADS=1        SIDE=X
    |_15_|        |_HELLO_|     |_1_|          |_X_|
      A               B$        HEADS          SIDE
```

The computer will file each of these variables
away in a box with that label. The contents of the box
can be changed any time we tell the computer that the
variable has changed. For example, HEADS=2 and
B$="GOODBYE" will switch the contents of those boxes
from 1 to 2 and from hello to goodbye.

3. Commands for variable operation include:
LET. This stores numbers or letters into a
(VARIABLE). You may see in a program: LET X=100 or
LET X$="SHELLY". In ATARI BASIC the (LET)
command is usually eliminated because the computer
automatically assumes its existence.

Note: normally reserved words in BASIC, such things as labels, commands and statements can be used on the ATARI if the LET statement is used. This is the only time that LET is really necessary. Words like: FOR, NEXT, DIM, LET, PRINT or even RUN can then be used as variables.

4. The specifications for variable labels include:

   Assign (IDENTIFIERS) like A, B. C3, DE49
   to (NUMERIC) variables. Assign identifiers
   like A$, B2$, ERROR$
   to (STRING) variables.

Type the following and write the results:

```
  LET HEADS = 1
  LET HEADS = 2
  PRINT HEADS             RESULTS (2)

TYPE and RUN:             PURPOSE:

  10 HEADS = 2            (ASSIGNS 2 TO VARIABLE HEADS)
  20 TAILS = 3            (ASSIGNS 3 TO VARIABLE TAILS)
  30 TOSS = HEADS+TAILS   (ASSIGNS SUM 2+3 TO VARIABLE TOSS)
  40 ? TOSS               (DISPLAYS VALUE OF VARIABLE TOSS)

  RESULTS: (5)
TYPE, NEW:

  10 ? "WHAT IS THE PERIMETER OF A SQUARE WITH A SIDE OF 5?"
  20 SIDE=5
  30 PERIMETER=SIDE*4
  40 ? PERIMETER

RESULTS: (20)
```

Note: ATARI will read and recognize variables up to (255) characters long. This contrasts with other micros which only sort through 1 or possibly 2 letters. This is why we have been able to use (words) rather than just letters for variables.

Example: if we could only look at the first 2 letters, the words APE and APPLE would look the same and the computer would pick the first one it came to.

We need some rules for numeric variables:

1. Each variable must include only (ALPHA) or (NUMERIC) characters.
2. The first must be an (ALPHA) character.
3. No (PUNCTUATION) or (SPACES) may be included.

Let's look at some variables. Are these valid or invalid on the ATARI? If not valid, why?

	VARIABLE	VALID	REASON
A.	A?	(NO)	(? NOT ALPHA OR NUMERIC)
B.	BALL	(YES)	(OK)
C.	2BAD	(NO)	(ALPHA MUST COME FIRST)
D.	P*3	(NO)	(* NOT ALPHA OR NUMERIC)
E.	NEXT	(YES)	(RESERVED, REQUIRES LET)

Data which contains both numbers and strings is called (ALPHANUMERIC) data. A (STRING VARIABLE) is used to name a memory location in which a string is stored.

Note: a string variable may be any valid numeric variable followed by a ($, DOLLAR SIGN).

Remember: In a LET or PRINT statement the data must be enclosed in (QUOTATION MARKS), however this is not necessary when the string is in a (DATA) statement.

Dimensioning:

Some BASICS allow limited strings to be used without reserving special memory locations. This is not the case with the ATARI. All STRINGS must have memory locations set aside. This procedure is called DIMensioning.

The statement format is:

DIM A$(X). DIM signifies there is a maximum number of characters allowed into variable (A$). It sets aside space, (X) in the computers memory to place data for that variable. Again it is required with all STRING variables, but not with numeric ones.

Examples: The statement is DIM A$(10) or DIM NAME$(25).

All strings to be used in a program may be dimensioned at one time, DIM A$(10), B$(20), C$(30), etc.

Note: A string may NOT be (DIMENSIONED) more than once in a program, so care must be taken to avoid placing DIMENSION statements inside of any (LOOPS).

TYPE and RUN:

```
  10 DIM MUSIC$(5)
  20 MUSIC$="MUSIC"
  30 ? MUSIC$                RESULTS: (MUSIC).
```

Now change line 10 so that it reads:

```
  10 DIM MUSIC$(4)
```

RUN the program again.   RESULTS: (MUSI).

Try the following program using DIM:

```
  10 DIM WHO$(20), DID$(20), WHAT$(20),
  20 LET WHAT$="A LITTLE LAMB"
  30 LET WHO$="MARY "
  40 LET DID$="HAD "
  50 ? WHO$;
  70 ? DID$;
  80 ? WHAT$;
```

RESULTS: (MARY HAD A LITTLE LAMB).

Type and RUN of the following program, indicating the purposes of each statement and the results.

```
  PROGRAM LINE          FUNCTION
  10 DIM NAME$(20)        (RESERVES 20 SPACES IN MEMORY)
  20 LET NAME$="MARY"     (ASSIGNS MARY TO NAME$)
  30 ? NAME$              (DISPLAYS MARY)
```

TYPE and RUN:

```
  10 DIM NAME$(20)
  20 ? "TYPE YOUR NAME?"
  30 INPUT NAME$
  40 ? "HELLO ";
```

What does the ";" do here?   (PUTS NAME ON SAME LINE).

```
  50 ? NAME$          RESULTS: (HELLO and NAME).
```

Subscripted Variables and Arrays:

NOTE: To avoid confusion with the parentheses used in the programming functions, this will be the form for <ANSWERS> for the remainder of this workbook.

As we have seen, simple BASIC variables come in the form: P, R, P1, P2, etc. We are now going to expand the use of variables through the use of SUBSCRIPTS.

A SUBSCRIPTED variable is written: P(5), P is the <VARIABLE>, 5 is the <SUBSCRIPT>

It consists of any letter followed by a <NUMBER> in <PARENTHESES>.

Which are the subscripted variables: X(1), C(3).

```
  X(1)    X    X1    C(3)    D
```

Notice that X, X1 and X(1) are three distinct variables. All three can appear in the same program. The computer will recognize them as three distinct variables.

Subscripted variables (like the simple variables we have been using) name a <MEMORY LOCATION> inside the computer. You can think of it as a box, a place to store a number. ASSIGN the following to the correct memory boxes below:

  P(2)=36   P(3)=12   P(4)=P(2)+P(3)

```
|____|   |____|   |_36_|   |_12_|   |_48_|
 P(0)     P(1)     P(2)     P(3)     P(4)
```

A set of subscripted variables is also called an <ARRAY>. This set of subscripted variables is a <ONE DIMENSIONAL ARRAY>, also known as a <VECTOR>. Subscripts may also include variables, Y(J) has J for a subscript:

  If J=1 then Y(J) = <Y(1)>.

  If J = 2 then Y(J) = <Y(2)>.

Subscripts can be more complex. Y(A+1), Y(B*2), etc., are all legal.

You must DIMension subscripted variables for the <MAXIMUM NUMBER> of values to be assigned to a particular subscripted variable. The DIM statement(s) in a program must be placed so that they are executed <BEFORE> the subscripted variable(s) are actually used in the program. The format for DIMensioning an array is similar to DIMensioning a string variable. In the example below X is the <VARIABLE> and the number 100 represents the <SUBSCRIPT>.

```
  100 DIM X(100)
```

Write a DIMension statement for variable A, with a maximum subscript of 50:

```
  100 <DIM A(50)>
```

One common use of subscripted variables is to store a list of numbers entered via <INPUT> or <READ STATEMENTS>. This can be done with a <FOR/NEXT> loop.

ATARI BASIC requires a modification of standard BASIC programs when assigning values to subscripted variables through READ or INPUT statements.

```
COMPARE:         STANDARD BASIC    ATARI BASIC
INPUT            INPUT A(I)        INPUT A
                                   A(I)=A
READ             READ A(I)         READ A
                                   A(I)=A
```

TYPE and RUN:

```
  PROGRAM                    PREDICTED OUTPUT
  10 DIM A (5)               <A(1)=9>
  20 FOR I = 1 TO 5          <A(2)=8>
  30 READ A                  <A(3)=7>
  40 A(I)=A                  <A(4)=6>
  50 NEXT I                  <A(5)=5>
  60 FOR I = 1 TO 5
  70 ? "A(";I;")=";A(I)
  80 NEXT I
  90 DATA 9,8,7,6,5
```

CHANGE line 30 and delete 90 to allow INPUT. <30 INPUT A>

RUN your program and note the results.

These programs used the control variable as the variable for the subscript, causing the subscript to <INCREASE BY ONE> each time through the loop. For another illustration, we will turn to an expensive Adding Machine.

Type in the following program and RUN it. INPUT 5 when asked How Many Numbers?. INPUT any number of your choice when asked A=?.

```
    PROGRAM
100 REM ***AN EXPENSIVE***
105 REM ***ADDING MACHINE
110 DIM A(10)
120 ? "HOW MANY NUMBERS";
130 INPUT NUMBER
140 ?
150 FOR I=1 TO NUMBER
160 ? "A=";
170 INPUT A
180 A(I)=A
190 NEXT I
200 T=0
210 FOR I=1 TO N
220 LET T=T+A(I)
230 NEXT I
240 ? "THE TOTAL IS "; T
```

For the RUN we just did, N was <5>. Therefore, <5> numbers will be entered by the operator and stored as A(1) through <A(5)>.

Remember in lines 170 and 180, first the input value is assigned to the simple numeric variable <A>. Then the subscripted variable <A(I)> is assigned the <SAME> value by letting <A(I)=A>.

Suppose the computer is running the program, it has just completed the FOR/NEXT loop in lines 150 to 190. The numbers entered by the user are now stored as follows: N=5, A(1)=37, A(2)=23, A(3)=46, A(4)=78, and A(5)=59. The computer is ready to proceed with line 200. This statement initializes the variable T, that is, <ASSIGNS> T its first value. Show the value of T after line 200 has been executed. T = <0>, Next the computer will execute the FOR/NEXT loop in lines 210 to 230. How many times will the FOR/NEXT loop be executed? <5>

WHY? <THE UPPER LIMIT OF THE FOR/NEXT LOOP IS N WHICH EQUALS 5>. The first time I will = 1, then I=2, I=3, I=4, and finally for I=5. Looking at line 220, we see: LET T=T+A(I). This statement tells the computer to add the values of A(I) to the <OLD> value of T and then assign the results as the <NEW> value of T. What is the value of T after line 220 has been executed for A(1), T= <37>, A(2), T=<60>, A(3), T=<106>, A(4), T=<184>, and A(5), T=<234>

Doubly-subscripted Variables:

Double subscripts are used with <ARRAYS> which might require several <COLUMNS> and <ROWS>. A rectangular arrangement of doubly-subscripted variables is called a <TABLE>, <MATRIX>, or a <TWO-DIMENSIONAL ARRAY>. Earlier we were describing arrays of singly subscripted variables which are also called <LISTS>, <VECTORS>, or <ONE-DIMENSIONAL ARRAYS>.

T(3) is a subscripted variable with <ONE> subscript.

T(7,5) has <TWO> subscripts which are separated by a <COMMA>.

Variables with <DOUBLE> subscripts arranged in an array of rows and columns is shown below. Put the following in the correct boxes below:

```
  LET A(1,3)=0
  LET A(2,1)=73
  LET A(1,1)=49
  LET A(2,3)=A(2,1)-A(1,1)
  LET A(1,2)=2*A(2,1)
  LET A(2,2)=INT(A(2,1)/A(2,3))
```

```
                   COLUMN 1        COLUMN 2           COLUMN 3
ROW 1     A(1,1) |_49_|  A(1,2) |_146_|    A(1,3) |_0__|
ROW 2     A(2,1) |_73_|  A(2,2) |_3__|     A(2,3) |_24_|
```

The above array has <TWO> rows and <THREE> columns.

With the arrangement shown above, we can relate subscripts to particular places <LOCATIONS, or "BOXES" for VALUES> in rows and columns. These are called the <ELEMENTS> in the array. These doubly subscripted variables are simply the names of a location in the computer. As with any other variable, you can think of them as the names for a place to store a number.

In the above example: A(2,3) is in row <2>, column <3>.

See if you can write a program to create an array of your class schedule, or your last report card.

LESSON VII:

COMPUTER FUNCTIONS:

Purpose: The special functions programmed into BASIC make it possible to accomplish many interesting things, in this lesson we will look at some of those functions, and what can be done with them.

MathematicaL Functions:

Most of the following functions are illustrated by simple <IMMEDIATE MODE> commands. Type them in to see how they work, then write the answer in the space provided. A few have short programs to type in and RUN.

RANDOM we will go into in more detail than the others because it is used so often by BASIC programmers.

RANDOM: <RND(X)> where "X" is a "dummy" (one without a value) variable, gives a random number between 0 and 1. It is used very frequently in programs to generate a set of random numbers between 0 and 1, 1 and 10, 1 and 100, etc.

```
   10 PRINT RND(1)
   20 GOTO 10
```

RUN this and then hit BREAK. What kinds of numbers do you get? Give examples: <0.0123456789> or <0.123456789>. In most cases, we would prefer whole numbers rather than decimals. So, let's combine the RANDOM with the INTEGER function and see what happens. But first a look at the

INTEGER function.

INTEGER: <INT(X)> truncates a decimal number at the decimal point, giving only the <WHOLE> number value. For a <NEGATIVE> it gives the lower value, for example:

```
TYPE these:                 RESULTS

    ?INT(2.8)               <2>
    ?INT(-2.8)              <-3>
    ?INT(1)                 <1>
    ?INT(14.7265)           <14>
```

TYPE and RUN:

```
    10 ? "TYPE A WHOLE NUMBER"
    20 INPUT N
    30 IF N/2=INT(N/2) THEN ? "EVEN"
    40 IF N/2<>INT(N/2) THEN ? "ODD"
    50 GOTO 10
```

Try a variety of numbers. Does it work? <YES>.

Now try INTEGER and RANDOM, TYPE these:

```
    10 ? INT(RND(1))
    20 GOTO 10
```

What kinds of numbers do you get now? <JUST 0's>. What we did was to eliminate the <DECIMAL> part and PRINT the <INTEGER>, which in each instance here was <0>. What we were getting was a number between 0 and 1, rounded down. In order to get a number up to 10 we will need to multiply our RND(1) by <10>.

TYPE this new line, and RUN:

```
    10 ? INT(RND(1)*10)
```

Now, what appears to be our range? <0 TO 9>. Did we get any zeros? <YES>. Did we get any 10's? <NO>. To get a range from 1 to 10 we must <ADD 1> to our program.

```
    10 ? INT(RND(1)*10)+1
```

Finally - we got to our range of a 1 to 10. What should we do if we wish to get a number range of 1 to 100?

```
    10 ? INT(RND(1) __(*100)+1)__
```

How would we get the numbers from 1 to 6 to simulate the roll of a dice?

```
    10 ? INT(RND(1) __(*6)+1)__
```

The even numbers from 10 to 20?

```
    10 ? INT(RND(1) __(*6)*2+10)__.
```

Let's look at some other functions:

ABSOLUTE VALUE: <ABS(X)> gives you the "ABSOLUTE" value of a number, which is the number without the + or - sign.

```
EXAMPLE:                 RESULTS

    ?ABS(5)              <5>
    ?ABS(-17)              <17>
    10 X=-8
    20 ? ABS(X)          <8>
```

This function is used to tell how far something is from <ZERO> when the sign is not important, or if a <NEGATIVE> sign would create an undiserable result.

SQUARE ROOT: <SQR(X)> gives you the square root of any positive expression.

```
TYPE:              RESULT

  ?SQR(25)           <5>
  ?SQR(144)          <12>
  ?SQR(2)            <1.41421356>
  ?SQR(-4)           <ERROR>
```

We have looked at the command to check our memory before.

FREE MEMORY: <FRE(X)> where X is a "dummy" variable, we usually use 1 or 0. It tells how much free <RAM> memory is available. You should check this whenever using a different computer, so that you will know how many "K" it has and therefore what software you can run on it, etc.

TYPE: ?FRE(0) WRITE RESULTS <ANSWERS WILL VARY>.

TYPE: in this program and RUN it.

```
  5 GR. 0
  10 ?:?
  20 ? "NUMBER","SQUARE","SQUARE ROOT"
  30 ?
  40 FOR NUMBER = 1 TO 10
  50 LET SQUARE=NUMBER*NUMBER
  60 LET ROOT=SQR(NUMBER)
  70 ? NUMBER,SQUARE,ROOT
  80 NEXT NUMBER
  90 ?
  100 ? "THIS PROGRAM LEFT YOU WITH "
  110 ? FRE(0); " MEMORY UNITS"
```

How much memory did this program take to RUN? Compare with previous FRE(0). <SHOW THE DIFFERENCE>

SIGN: <SGN(X)> is used to evaluate an expression for its sign. Returns <-1> if it evaluates to a negative number, <0> if zero, and <1> if positive number.

TYPE:	RESULTS
?SGN(5*12-15)	<1>
?SGN(25-50)	<-1>
?SGN(5-5)	<0>

LOG: <LOG(X)> returns the natural logarithm of a positive number in the parentheses.

TYPE:	RESULTS
?LOG(1)	<0>
?LOG(100)	<4.605...>
?LOG(1.5)	<0.405...>

COSINE: <COS(X)> returns the trigonometric cosine of the expression.

TYPE:	RESULTS
?COS(0)	<1>
?COS(27)	<0.292...>

SINE: <SIN(X)> returns the trigonometric sine of the expression in the parentheses.

TYPE:	RESULTS
?SIN(0)	<0>
?SIN(1.7)	<0.991...>

The following is a list of the functions and statements described in the ATARI BASIC REFERENCE MANUAL, starting on page 33. We have covered some of the more important functions and will deal with any others as they apply to the material we are working with.

ABS	LOG	ATN	ADR
CLOG	RND	COS	FRE
EXP	SGN	SIN	PEEK
INT	SQR	DEG/RAD	POKE
USR			

STRINGS and COMPUTER FUNCTIONS:

Storing Strings: You need to understand how characters are stored in the computer's memory. Computer memory can store <NUMBERS>, but not <CHARACTERS>. Therefore, characters must be converted to numbers. The special numeric code is called ASCII (American Standard Code for Information Interchange). The ATARI computer uses a slightly different code, called ATASCII (ATARI ASCII). This involves a simple substitution of the code number for the letter. A is 65, B is 66, C is 67, and so on.

ASCII function: <ASC(A$)> converts the first character of a string to its ATASCII code. To see how this works, try the following program:

```
  10 DIM A$(1)
  20 ? "ENTER ONE CHARACTER";
  30 INPUT A$
  40 ? "THE ATASCII CODE"
  45 ? "FOR ";A$;" IS:"
  50 ? ASC(A$)
  60 GOTO 20
```

(For the complete ATASCII CODE listing see the ATARI BASIC REFERENCE MANUAL APPENDIX C)

Try the following characters and enter their ATASCII codes below:

```
  A = <65>    a = <47>    D = <68>    d = <100>
  9 = <57>    ? = <63>    * = <42>    S = <83>
```

To reverse the above procedure the CHR$ function: <CHR$(N)> will give the character or command when you indicate the ATASCII number.

Redo the above program, changing from string variables to numeric variables and changing lines 40 and 50. Eliminate line 10.

```
  20 ? "ENTER A NUMBER FROM 0 TO 255"
  30 INPUT A
  40 ? "THE CHARACTER OR COMMAND FOR ";A;" IS: ";
  50 ? CHR$(A)
  60 GOTO 20
```

Enter the character or function for the following:

```
  38 = <&>          125 = <CLEARS SCREEN>
  253 = <BUZZER>    113 = <q>
```

Sometimes it is necessary to know the length of a specified string.

LENGTH: <LEN(A$)> gives the length of the specified string, including all spaces.

TYPE:

```
  WORD$="A SHORT PHRASE"
  ?LEN(WORD$)= <14>
```

SUBSTRINGS and COMPUTER FUNCTIONS:

Substrings: To use only a part of a string variable the following are possible techniques.

Note: The ATARI doesn't use LEFT$, MID$, AND RIGHT$ functions as other BASICs do.

```
TYPE:                        RESULT

    10 DIM A$(10)
    20 A$="ABCDEFGHIJ"
    30 ? A$(3)                   <CDEFGHIJ>
    40 ? A$(1,3)                 <ABC>
    50 ? A$(2,5)                 <BCDE>
    60 ? A$(LEN(A$)-3)       <GHIJ>
```

1. A$(3) will specify everything from the <3RD> character on.

Note: to get (X) characters from the end of a string use: A$(LEN(A$)-X), where X is one less than the number you want, if X=3 your will get <GHIJ>, because LEN(A$)=10 and

10-3 = 7, so we are really saying A$(7).

2. A$(1,3) specifies the characters from the beginning to the <3RD> character, or the 1st <3> characters.

3. A$(2,5) specifies the characters from the <2ND to the 4TH>.

STRING CONCATENATION: Allows you to join strings or substrings together to form new longer strings. A string can be any length up to the end of the available <RAM>. Fill in the boxes below if C$=HER, A$=TO, AND B$=GET

```
    STRING 1     +     STRING 2     +     STRING 3
    |_TO_|       +     |_GET_|      +     |_HER_|
     A$                  B$                 C$
Becomes:         $1  $2  $3
               |_TO|GET|HER_|
    ?A$;B$;C$ = __(TOGETHER)__.
```

In ATARI BASIC the second and third strings are concatenated by making them substrings which start just after the last character of the string before it.

EXAMPLE: STRING 1 = STRING 1 + STRING 2 would be A$(LEN(A$)+1)=B$ STRING 3 = STRING 1 + STRING 2 would be C$=A$ C$(LEN(C$)+1)=B$

Enter the following in Direct Mode:

```
  DIM WORD$(40)
  LET WORD$="MARY HAD A LITTLE LAMB"
```

Now experiment with printing out pieces of the words in WORD$ by using the following kinds of instructions:

```
TYPE:                 RESULTS

  ? WORD$(6)          <HAD A LITTLE LAMB>
  ? WORD$(10,10)        <A>
  ? WORD$(12,17)        <LITTLE>
```

Now TYPE and RUN:

```
  5 DIM WORD$(40)
  10 GR. 0
  20 ?
  30 ? "TYPE IN WORD"
  40 INPUT WORD$
  50 ? WORD$;" CONTAINS ";LEN(WORD$);" LETTERS."
  60 ?
  70 GOTO 20
```

RESULTS: <WILL GIVE THE LENGTH OF THE TYPED WORD>.

TYPE and RUN:

```
10 DIM A$ (50)
20 ? "TYPE NAME"
30 INPUT A$
40 ?:?
50 FOR N = LEN(A$) TO 1 STEP -1
60 ?A$(N,N);
70 NEXT N
80 ?:?
90 GOTO 20
```

RESULTS: <WILL DISPLAY NAME IN REVERSE ORDER>.

EXPERIMENT: with this program and have fun.

LESSON VIII:
Sound, Color, and Graphics

Purpose: We have used some short programs with sound, color and graphics to illustrate programming techniques, now in this lesson we will look at the specific commands that make these possible.

SOUND:

In ATARI BASIC, SOUND statements control the <AUDIO ON THE T.V. OR MONITOR>. The command is:

SOUND V,P,D,L or SO. V,P,D,L:

V= <VOICE>:

The ATARI computer has <4> independent voices or sound registers. This means it can make as many as <4> different sounds simultaneously. The different voices blend together in the television speaker, like voices in a chorus. The voices are numbered <0> through <3>. You must use a <SEPARATE> SOUND statement to control each voice.

RHYTHM is controlled by <DELAYS> built into the program or through DATA statements.

P= <PITCH>:

or NOTE, the second number sets the <PITCH>, it can be any number between <0> and <255>. The lowest notes are played by the <LARGEST> numbers so they seem to be opposite from what you might guess. The ATARI computer can produce all notes: sharps, flats, and neutrals, from <ONE OCTAVE> below middle C to <TWO OCTAVES> above it. It can produce a good many other tones as well. For example, there are (6) intermediate values between middle C and the tone one-half step below it, B. Such tones do not match any of the notes on the regular "Western" music scale, so they will be of no use for programming music, unless you want to try some "Eastern" or Oriental music. You can, however, use these tones for sound effects.

D= <DISTORTION>:

The ATARI computer produces both <PURE> and <DISTORTED> tones.

This, the third number in a SOUND statement regulates the <DISTORTION>. It can be any value between <0> and <15>. Distortion values of <10 and 14> generate pure tone. Other even numbered distortion values <0,2,4,6,8,and 12> introduce different amounts of noise into the pure tone. The amount of noise depends on both the distortion value and the pitch value.

L= <LOUDNESS>:

The fourth number in a SOUND statement controls the <VOLUME> of the specified voice. "L" is used because "V" was used for <VOICE> It lets the program determine the <AUDIO> level. It also allows the program to mix multiple-voices, with each voice at a different volume level. The loudness value can be between <0=SILENT> and <15=LOUDEST>. Loudness is direct, 8 produces a sound half as loud as 15 and 12 is halfway between 8 and 15 in loudness.

Note: to turn off the sound when using the direct mode you must type <END>. BREAK will <NOT> turn it off.

Type in direct mode: SO. 0,96,10,8 and hit <RETURN>. Once turned on, each sound generator stays on until the program reaches an <END> statement or the program shuts it off, usually with the command SO. 0,0,0,0.

TYPE: SO. 1,121,10,8 <R>. This plays middle C on sound register 0. On the line below the one playing middle C, type: SO. 0,96,10,8 <R>, now take your cursor up to the first of the two commands and press <R> 2 times. What did the computer do? <SOUNDED ONE NOTE THEN THE NEXT>.

Now type the following 1 line command: SO. 0,108,10,8:SO. 1,96,10,8 and write the results: (TWO NOTES PLAY AT THE SAME TIME)

The notes you are playing are E and C. ADD G <81> and a higher C <60> to your line and play a "C" chord.

REMEMBER: you can use <4> voices at one time. How should this command be written? <SO.0,100,10,8:SO.1,96,10.8:ETC.>

The following program shows off the ATARI's complete tonal range.

```
  10 FOR TONE=-255 TO 255
  20 SO. 0,ABS(TONE),10,8
  30 ? "NOTE VALUE: ";ABS(TONE)
  40 FOR TIME=1 TO 50:NEXT TIME
  50 NEXT TONE
  60 SO. 0,0,0,0
```

As you listen, notice that the low notes seem to last longer than the high notes. You can see that the program holds each tone for the same length of time (line 30). But the tone produced by a pitch value of 255 is very nearly the same as that produced by 254, 253, and even 252. These low tones run together, sounding like <ONE LONG NOTE>. In contrast there is a distinct difference between pitch values <11 and 10>. Each change in pitch value is very obvious, after the program goes smoothly through the low tones, but ends up jumping roughly between the higher tones. The following program will play the notes of the scale as we are familiar with them.

```
10 READ A
20 IF SCALE=999 THEN END
30 SO. 0,SCALE,10,10
40 FOR TIME=1 TO 400:NEXT TIME
50 ? SCALE
60 GOTO 10
70 END
80 DATA 29,31,35,40,45,47,53,60,64,72,81,91,96,108,121
90 DATA 128,144,162,182,193,217,243,999
```

NOTE: The DATA statement in line 80 ends with a 999, which is outside of the designated range. The 999 is used as an <END OF DATA> marker. This program will tell the computer to continue to READ and play notes until that marker is reached. It will allow you to program as many notes as you wish without getting an (OUT OF DATA) ERROR. You can experiment all you want if you will be sure to start a new DATA line whenever the one you are on reaches 3 computer <LOGICAL> lines, then you place 999 or some other number out of the tonal range, as your last DATA number to tell the computer that you are finished.

Generally speaking, odd numbered distortion values

<1,3,5,7,9,11,,13,15> silence a specified voice. But if the voice is off, a SOUND statement with an odd-numbered distortion value causes a single click, then silence. Turning the voice off causes another click. Try this program to see how odd numbered distortion values work:

```
10 FOR J=1 TO 20
20 SO. 0,0,1,8
25 FOR K=1TO25:NEXT K
30 SO. 0,0,0,0
40 FOR K=1 TO 100:NEXT K
50 NEXT J
```

Experiment with a DELAY LOOP to adjust the speed of the clicks.

How about a metronome for piano practice? What happens if you change line 20 to SO. 0,96,1,8 <STILL CLICKS, NO SOUND> or to SO. 0,96,2,8 <DISTORTED SOUND> or to SO. 0,96,10,8 <ON and OFF NOTE>.

Type in direct mode a SOUND command with a loudness of 8, play it, then change to 15, 3, and 0 respectively. Can you hear a loudness of 1? <DEPENDS ON VOLUME SETTING, etc.>

SOUND EFFECTS:

Let's see how some sound effects are created.

```
10 REM ** BOUNCING BALL **
20 FOR N=25 TO 1 STEP -1
30 FOR K=1 TO 5
40 SO. 0,125,14,6
50 NEXT K
60 SO. 0,0,0,0
70 FOR K=1 TO N*5:NEXT K
80 NEXT J
100 END
```

TRY THIS ONE: What does it sound like? Notice how the program has been compressed.

```
  10 V=64:FOR M=1TO30:SO.0,V-J,10,10
  20 SO.1,F+J,10,10:FOR K=1TO 30-J:NEXT K
  30 SO.0,0,0,0:SO.1,0,0,0:FOR K=1TO10:NEXT K
  40 NEXTJ:END
```

Try making some changes in the variables and re-RUNning the program.

More Sounds:

```
  5 REM **STAR WARS**
 10 FOR SD=1 TO 8:FOR SX=0 TO 2:SO. 1,12,4,8
 20 FOR SY=20 TO 160 STEP 8:SO. 0,SY,10,8*(SX>0)
 25 NEXT SY:NEXT SX
 30 NEXT SD:GOTO 100
100 SO. 0,0,0,0:SO. 1,0,0,0

5 REM **DOOR BELL**
10 FOR SD=1 TO 6:FOR SY=14 TO 0 STEP -0.35:SO. 0,255,10,SY
20 SO. 1,252,10,SY:NEXT SY:NEXT SD
```

By changing the data statement in the following, you can get it to play one voice of any piece of music.

The following is a listing of the approximate note values to put into the "P" (PITCH) variable.

G=162, G#=153, A=144, A#=136, B=128 (middle)C=121, C#=114

D=108, D#=102, E=96, F=91, F#=85, G=83, G#=76, A=72

The time each note is held is controlled by the delay loop, give a 1/4 note a value of 25, 1/2 note is 50, a whole note is 100. Alternate the note or PITCH value and the DELAY value in the data statement.

TYPE and RUN:

```
  10 READ N,D
  20 SO,1,N,10,8
  30 FOR I=1TOD:NEXT 1
  40 SO.0,0,0,0
  50 IF N=999 THEN END
  60 GOTO 10
  70 DATA 40,25,31,25,29,25,53,100,40,25,31,25,
     29,25,53,100,999,0
```

See if you can re-write the program to play a tune with several voices.

INTRODUCTION TO GRAPHICS:

Next we will look at the ATARI graphics. This is another area where you can be very creative. As you go through these programs try mixing SOUND and GRAPHICS.

Color Registers:

The ATARI computer has five color registers numbered <0> through <4>. Each color register has a specific application; for example, Color Register 4 changes the color of the <FRAME AROUND THE SCREEN> and Color Register 2 changes the <BACKGROUND> color of the <SCREEN> itself. Two numbers, the color number and its shade value, are associated with each color register.

SETCOLOR:

Use the SETCOLOR (SE.) statement to assign specific colors and associated shades to each color register. The structure of the SETCOLOR statement is as follows:

```
   SE. X,Y,Z  where:
       X <0 to 4> represents the (COLOR REGISTER).
       Y <0 to 15> represents the COLOR <NUMBER>
       Z <0 to 14> represents the <LUMINANCE> number.
```

COLOR NUMBERS	APPROXIMATE COLOR ON TV SCREEN
0	Gray
1	Gold
2	Orange
3	Red-orange
4	Pink
5	Purple or Violet

6	Red-blue
7	Blue
8	Blue
9	Light Blue
10	Turquoise
11	Green-blue
12	Green
13	Yellow-green
14	Orange-green
15	Light Orange

LUMINANCE:

Luminance, (BRIGHTNESS) is changed on every (EVEN) number: 0,2,4,6,8,10,12&14.

Note: #10 will cause a blank screen, either reassign the number or press <RETURN>.

(VARIABLES) can be substituted for numbers in the SETCOLOR statement. An example of this follows.

Practice: In direct command mode use Color Register 2 or 4 and try various colors and shades. Example: SE. 2,3,6.

TYPE NEW, and then the following program:

```
10 REM SHOWS ALL COLORS AND SHADES
20 REM USES COLOR REGISTER 2
30 FOR X=0 TO 15
40 FOR Y=0 TO 14
50 SE. 2,X,Y
55 FOR I=0 TO 100: NEXT I
60 NEXT Y: NEXT X: END
```

Experiment with SETCOLOR on Register 1 to determine the shade of the letters on the screen.

GRAPHICS:

TYPE NEW, and the following program:

```
10 REM ****BLAST-OFF***
20 SE. 2,0,2
30 FOR T=10 TO 0 STEP -1
40 ? T
```

```
50 FOR D=1 TO 100: NEXT D
60 NEXT T
65 SE. 2,4,14
70 ? CHR$(253);"BLAST-OFF!"
80 SE. 2,0,2: END
```

GRAPHICS MODES:

Let's take a look at some of the graphics modes available.

We already know that (GRAPHICS 0 or GR.0) will clear the screen. What else does it do?

COMPUTER INSTRUCTION	OBJECT ON SCREEN	SIZE (NO. OF OBJECTS ON SCREEN)
GR. 0	regular type 1 color, 2 luminances)	24 lines of 40 char.
GR. 1	large type (double width) (5 colors)	20 lines of 20 char.
GR. 2	large type (double and width) (5 colors)	10 lines of 20 char.
GR. 3	large graphics squares (4 colors)	20 lines of 40 squares
GR. 4	smaller graphics points (2 colors)	40 lines of 80 squares
GR. 5	smaller graphics points (4 colors)	40 lines of 80 squares
GR. 6	very small graphics points (2 colors)	80 lines of 160 points
GR. 7	very small graphics points (4 colors)	80 lines of 160 points
GR. 8	high-resolution graphics (1 color, 2 luminances	160 lines of 320 points

Note: check ATARI BASIC manual for Graphics Modes 9 - 11.

DEFAULT COLORS:
In those Graphics Modes where large text is available through the PRINT #6 command if no (SETCOLOR) commands are given before the PRINT #6 instructions are encountered, the text will appear in default colors in the (COLOR REGISTER).

COLOR REGISTER	CHARACTER COLORED	DEFAULT COLOR
0	Capital Letters	Orange
1	Lowercase	Light Green
2	Inverse Capitals	Dark Blue
3	Inverse Lowercase	Red
4	Screen Background	Dark Gray, Black

Enter Graphics Mode 1 by typing (GR.1). Type in various PRINT #6 instructions to see what appears on the screen. Do the same with Graphics 2. To get back into the regular (TEXT) mode, TYPE GR.0, or use the <RESET> key.

TYPE (exactly) and RUN:

```
10 GR.1:REM USE INVERSE FOR 3 AND 4 BELOW:
20 ? #6;"ONE two THREE four"
30 END
```

List the colors for each word:

1. (ORANGE), 2. (LIGHT GREEN), 3. (DARK BLUE), 4. (RED).

Change to GR.2 in line 10, RUN: what happened? (CHARACTERS GOT LARGER).

Now TYPE in the following SETCOLOR commands after the above program has RUN. Do this while still in graphics mode 2, using the Direct Command mode. Observe what happens on the screen to the results of the previously RUN program.

```
LINE                 RESULT
SE. 0,8,0            (ONE TURNS DARK BLUE)
SE. 1,2,10           (TWO TURNS GOLD)
SE. 2,15,14          (THREE TURNS LIGHT GOLD,
                        AS DOES TEXT WINDOW)
SE. 3,0,6            (FOUR TURNS GRAY)
SE. 4,4,2            (BACKGROUND TURNS BRIGHT RED)
```

Which thing did each change in color register affect?

LIST the above program and add the following lines. Then RUN again.

```
30 FOR X=0 TO 3:REM NUMBER OF (COLOR REGISTER)
40 FOR Y=0 TO 10:REM NUMBER OF (FLASHES)
50 SE. X,0,0
60 FOR TIME=1 TO 100:NEXT TIME:REM (DELAY)
70 SE. X,0,14
80 FOR TIME=1 TO 100:NEXT TIME:REM DELAY
90 NEXT Y: NEXT X
```

RUN the program, then press the RESET key and add the following line. Now RUN it again.

```
     15 SE. 4,4,10
```

NOTE: Whenever GR.0,1,or 2 is executed, the screen is (CLEARED) and default colors are placed back in the color registers.

Other Graphics Modes (3-8):

These modes are sometimes referred to as low resolution or LORES Graphics.

To see all the colors and shades available to you, type in the following program and RUN it. You can speed it up or slow it down after you have run it the first time.

```
     10 REM ***FLASHING COLORS***
     20 GR.0:FOR R= 0 TO 255
     30 POKE 710,R: FOR I= 1 TO 250 :NEXT I
     40 POS. 10,10:?" THIS IS COLOR ";INT(R/16)
     50 NEXT (R): END
```

Note: there are 2 new commands in this program, POS. we will look at later. POKE is a special command which can be used to send instructions directly to computer memory, to be acted upon immediately. In the program above it is used to put a color number into the memory location which controls colors. POKE has its counterpart, PEEK whick allows us to look at a memory location to determine what number is in that location. We can then use that information to create other effects.

PLOT and DRAWTO Commands:

We can put points of color or text at specific locations on the screen. The PLOT and DRAWTO commands make this possible.

The PLOT command is: (PLOT X,Y) where "X" represents the number of spaces across the screen, and "Y" is the number of spaces down the screen. Visualize a grid pattern on the screen. A (GRAPHICS) point is placed at the intersection of the X and Y coordinates given in the structure of the command. In Graphics Mode 0 and 3, coordinates can be given from 0 to 39 on the X axis and from 0 to 19 (0 to 23 in GR.0) of the Y axis. Try placing a few points on the screen using the Direct Command. For example: PLOT 4,8

This places a (HEART) at the given location on the screen.

To use GR. 3, first TYPE COLOR X (SEE TABLE), then try: PLOT 4,8

NOW try to put a point in each corner of the screen, and one in the center.

REMEMBER that the COLOR instruction does NOT place a color in any of the registers, but does tell the computer which register will be controlling the color of the graphics point.

Determine the missing instruction from the following program and write it in: (You do not need to type it.)

```
PROGRAM               MISSING
10 GR. 3              (LINE 15 OR 25 COLOR 1
20 SETCOLOR 3,9,4
30 PLOT 15,6
```

A COLOR command directly controls the choice of the color register involved in any following PLOT or DRAWTO commands. The following table illustrates this relationship. Example: COLOR 1 corresponds to Color Register 0, COLOR 2 to Color Register 1, COLOR 3 to Color Register 2, and COLOR 3 to Color Register 3.

Try this SAMPLE PROGRAM: using COLOR Command.

```
     10 GR. 3
     20 SE. 0,6,8
     30 SE. 2,0,6
     40 COLOR 1
     50 PLOT 15,15
```

BEFORE you RUN it, see if you can figure out what the colors will be (HINT: look at Color List).

Change the COLOR and SETCOLOR instruction to get different colored squares on the screen.

TYPE the following Program and RUN it:

```
     10 GR.3
     20 COLOR 1
     30 X=INT(RND(1)*40):Y=INT(RND(1)*20)
     40 PLOT X,Y
     50 GOTO 20
HIT <BREAK> and ADD:
     20 C=INT(RND(1)*4)
     25 COLOR C
```

Put in a FOR/NEXT loop to create some interesting effects.

DRAWTO X,Y:

To Draw a line, you must start from a PLOT point and DRAWTO a new point. Enter and RUN the following:

```
     10 REM FLASHING LIGHTNING
     20 GR. 3
     30 COLOR 1
     40 PLOT 10,0
     42 DRAWTO 20,19
```

After RUNning the program, add:

```
45 FOR X=1 TO 10
50 SE. 0,X,8
60 FOR D=1 TO 100: NEXT D
70 SE. 0,9,14
80 FOR D=1 TO 100:NEXT D
100 NEXT X
```

TRY changing the Graphics Mode to GR. 5 and RUN the program. Then try GR.7 and RUN it again.

Using only PLOT and DRAWTO, see if you can draw a square near the center of the screen. Draw it in Graphics 3 and make each side 25 units long. RUN your program.

Show it to the INSTRUCTOR before continuing.

The PLOT command puts points at the intersection of a grid position on the screen.

LIST your square program and change it to GR. 5. Now try to POSITION it back in the center of the screen. When you are sure you have succeeded in this, try GR. 7.

POSITION (POS.X,Y): is similar, except the coordinate would not be illuminated, rather the next PRINT statement to be executed would be located at that point. POSITION can also be used in Graphics Modes 0, 1, and 2 to move the cursor to different places. It is also used to place text on the screen at specific locations, it replaces, TAB, VTAB, and HTAB

Example: 10 POS. 0,0:?"*, will put an asterisk in the upper left corner of the screen. See if you can write a program to put a character in the upper right, lower left, etc.

All the Graphics we have done so far have had a (Text Window) at the bottom. On the 800 model, to remove it, just add 16 to the Graphics Mode number. It is usually listed as (NUMBER) + 16, but sometimes you will see it already added, as 17. On the XL model you will need to put a delay loop to hold it on the screen.

To see POSITION at work in GR.0, 1 AND 2, try the following:

```
     10 GR.0
     20 POS. 5,5
     30 ?"TEXT"
ALSO:10 GR. 2
     20 POS. 8,4
     30 ? #6;"text"
AND: 10 GR. 1
   20 POS.8,4:?#6;"TEXT"
```

Try changing the POSITION "coordinates" of line 20 in each of the programs, but remember your limitations from the "Graphics Mode" chart, otherwise you may "ERROR OUT" with the cursor out of range.

Happy PLOTing!!

SUMMARY

This workbook began with an introduction to the keyboard and screen editing. Lesson two introduced the concepts of BASIC programming statements and commands, then explored the treatment of mathematical functions on the computer. The third lesson introduced program planning and the use of computer peripherals, including the disk drive and the printer. In lesson four we expanded the understanding of programming features. Lesson five looked at the BASIC concepts of branching and loops. The treatment of variables in BASIC was developed in lesson six, while lesson seven explored functions programmed into the computer. In the concluding lesson, we were introduced to the use of sound, color, and graphics in programming. Throughout the workbook there was an attempt to allow the student to discover as many of the programming concepts as possible. You should now continue with studies of computers and computer programming. You can work on your own or select other courses as they are available.